42 C Beginner Exam Review:

```
======================================./0-0-
aff_a.txt=========================================
Assignment name  : aff_a
Expected files   : aff_a.c
Allowed functions: write
----------------------------------------------------------
----------------------

Write a program that takes a string, and displays the
first 'a' character it
encounters in it, followed by a newline. If there are no
'a' characters in the
string, the program just writes a newline. If the number
of parameters is not
1, the program displays 'a' followed by a newline.

Example:

$> ./aff_a "abc" | cat -e
a$
$> ./aff_a "dubO a POIL" | cat -e
a$
$> ./aff_a "zz sent le poney" | cat -e
$
$> ./aff_a | cat -e
a$
==========================================================
================================
```

```c
#include <unistd.h>

int       main(int argc, char *argv[])
{
     int       i;

     i = 0;
     if (argc != 2)
     {
          write(1, "a\n", 2);
          return (0);
     }
```

```
    else
    {
        while (argv[1][i])
        {
            if (argv[1][i] == 'a')
            {
                write(1, "a", 1);
                break ;
            }
            i += 1;
        }
        write(1, "\n", 1);
        return (0);
    }
}
```

```
======================================./0-0-
ft_countdown.txt=========================================
Assignment name  : ft_countdown
Expected files   : ft_countdown.c
Allowed functions: write
----------------------------------------------------------
----------------------

Write a program that displays all digits in descending
order, followed by a
newline.

Example:
$> ./ft_countdown | cat -e
9876543210$
$>
==========================================================
================================
```

```
#include <unistd.h>

int  main(void)
{
    write(1, "9876543210\n", 11);
}
```

```
======================================./0-0-
```

```
ft_print_numbers.txt=========================================
Assignment name  : ft_print_numbers
Expected files   : ft_print_numbers.c
Allowed functions: write
--------------------------------------------------------------------------------
```

Write a function that displays all digits in ascending order.

Your function must be declared as follows:

void ft_print_numbers(void);

```
==========================================================================================
```

```c
#include <unistd.h>

void ft_print_numbers(void)
{
	write(1, "0123456789\n", 10);
}
```

```
======================================./0-0-hello.txt=========================================
Assignment name  : hello
Expected files   : hello.c
Allowed functions: write
--------------------------------------------------------------------------------
```

Write a program that displays "Hello World!" followed by a \n.

Example:

```
$>./hello
Hello World!
$>./hello | cat -e
Hello World!$
```

```
$>
==========================================================
================================
```

```c
#include <unistd.h>

int       main(void)
{
     write(1, "Hello World!\n", 13);
     return (0);
}
```

```
======================================./0-0-
maff_alpha.txt=========================================
Assignment name  : maff_alpha
Expected files   : maff_alpha.c
Allowed functions: write
----------------------------------------------------------
----------------------
```

Write a program that displays the alphabet, with even
letters in uppercase, and
odd letters in lowercase, followed by a newline.

Example:

```
$> ./maff_alpha | cat -e
aBcDeFgHiJkLmNoPqRsTuVwXyZ$
==========================================================
================================
```

```c
#include <unistd.h>

int       main(void)
{
     write(1, "aBcDeFgHiJkLmNoPqRsTuVwXyZ\n", 27);
     return (0);
}
```

```
======================================./0-1-
aff_first_param.txt=======================================
==
Assignment name  : aff_first_param
Expected files   : aff_first_param.c
Allowed functions: write
-----------------------------------------------------------------
---------------------

Write a program that takes strings as arguments, and
displays its first
argument followed by a \n.

If the number of arguments is less than 1, the program
displays \n.

Example:

$> ./aff_first_param vincent mit "l'ane" dans un pre et
"s'en" vint | cat -e
vincent$
$> ./aff_first_param "j'aime le fromage de chevre" | cat
-e
j'aime le fromage de chevre$
$> ./aff_first_param
$
=================================================================
===============================
```

```c
#include <unistd.h>

int  main(int argc, char *argv[])
{
     int  i;

     i = 0;
     if (argc < 2)
     {
          write(1, "\n", 1);
     }
     else
     {
          while (argv[1][i])
          {
               write(1, &argv[1][i++], 1);
          }
          write(1, "\n", 1);
```

```
    }
    return (0);
}
```

```
======================================./0-1-
aff_last_param.txt========================================
=
Assignment name  : aff_last_param
Expected files   : aff_last_param.c
Allowed functions: write
----------------------------------------------------------
----------------------

Write a program that takes strings as arguments, and
displays its last
argument followed by a newline.

If the number of arguments is less than 1, the program
displays a newline.

Examples:

$> ./aff_last_param "zaz" "mange" "des" "chats" | cat -e
chats$
$> ./aff_last_param "j'aime le savon" | cat -e
j'aime le savon$
$> ./aff_last_param
$
==========================================================
================================
```

```
#include <unistd.h>

int     main(int argc, char *argv[])
{
    int  i;

    i = -1;
    if (argc > 1)
    {
        while (argv[argc - 1][++i])
        {
            write(1, &argv[argc - 1][i], 1);
```

```
          }
     }
     write(1, "\n", 1);
     return (0);
}
```

```
======================================./0-1-
maff_revalpha.txt=========================================
Assignment name  : maff_revalpha
Expected files   : maff_revalpha.c
Allowed functions: write
----------------------------------------------------------
----------------------

Write a program that displays the alphabet in reverse,
with even letters in
uppercase, and odd letters in lowercase, followed by a
newline.

Example:

$> ./maff_revalpha | cat -e
zYxWvUtSrQpOnMlKjIhGfEdCbA$
==========================================================
================================
```

```
#include <unistd.h>

int  main(int argc, char *argv[])
{
     write(1, "zYxWvUtSrQpOnMlKjIhGfEdCbA\n", 27);
}
```

```
======================================./0-1-
only_a.txt=========================================
Assignment name  : only_a
Expected files   : only_a.c
Allowed functions: write
----------------------------------------------------------
----------------------
```

```
Write a program that displays a 'a' character on the
standard output.
==========================================================
================================
```

```
#include <unistd.h>

int  main(void)
{
     write(1, "a", 1);
     return (0);
}
```

```
======================================./0-1-
only_z.txt=========================================
Assignment name  : only_z
Expected files   : only_z.c
Allowed functions: write
----------------------------------------------------------
----------------------

Write a program that displays a 'z' character on the
standard output.
==========================================================
================================
```

```
#include <unistd.h>

int  main(void)
{
     write(1, "z", 1);
}
```

```
======================================./0-2-
aff_z.txt=========================================
Assignment name  : aff_z
Expected files   : aff_z.c
Allowed functions: write
----------------------------------------------------------
```

```
----------------------
```

Write a program that takes a string, and displays the first 'z' character it encounters in it, followed by a newline. If there are no 'z' characters in the string, the program writes 'z' followed by a newline. If the number of parameters is not 1, the program displays 'z' followed by a newline.

Example:

```
$> ./aff_z "abc" | cat -e
z$
$> ./aff_z "dubO a POIL" | cat -e
z$
$> ./aff_z "zaz sent le poney" | cat -e
z$
$> ./aff_z | cat -e
z$
==========================================================================================
```

```c
#include <unistd.h>

int  main(int argc, char **argv)
{
     (void)argc;
     (void)argv;
     write(1, "z\n", 2);
     return (0);
}
```

```
======================================./1-0-ft_strcpy.txt=========================================
Assignment name  : ft_strcpy
Expected files   : ft_strcpy.c
Allowed functions:
--------------------------------------------------------------------------------
```

Reproduce the behavior of the function strcpy (man strcpy).

Your function must be declared as follows:

```
char    *ft_strcpy(char *s1, char *s2);
==========================================================
================================
```

```
char *ft_strcpy(char *s1, char *s2)
{
     while ((*s1++ = *s2++))
          ;
     return (s1);
}
```

=====

```
char *ft_strcpy(char *dest, char *src)
{
     int i;

     i = 0;
     while(src[i] != '\0')
     {
          dest[i] = src[i];
          i++;
     }
     dest[i] = '\0';
     return (dest);
}
```

===

```
#include <stdio.h>

char *ft_strcpy(char *dest, char *src);

int main(void)
{
     char hero[] = "pacman";
     char villian [] = "frogger";

     ft_strcpy(hero, villian);
     printf("%s\n", hero);
     return 0;
}

====
#include <stdio.h>

char *ft_strcpy(char *s1, char *s2);

int main(void)
{
    char boy[] = "harry";
    char girl[] = "sally";

    printf("boy is: %s\n", boy);
    printf("girl is: %s\n", girl);
    ft_strcpy(boy, girl);
    printf("boy is: %s\n", boy);
    return (0);
}
```

```
==========================================./1-0-
ft_strlen.txt==============================================
Assignment name  : ft_strlen
Expected files   : ft_strlen.c
Allowed functions:
-----------------------------------------------------------------
------------------------

Write a function that returns the length of a string.

Your function must be declared as follows:

int  ft_strlen(char *str);
=================================================================
====================================
```

```
int  ft_strlen(char *str)
{
     int i;

     i = 0;
     while (str[i])
          i++;
     return (i);
}
#include <stdio.h>

int ft_strlen(char *str);

int main(void)
{
    char monster[] = "ogrefly";
    printf("%s has %d characters", monster, ft_strlen(monster));
    return (0);
}
```

```
======================================./1-0-repeat_alpha.txt=========================================
Assignment name  : repeat_alpha
Expected files   : repeat_alpha.c
Allowed functions: write
--------------------------------------------------------------------------------
```

Write a program called repeat_alpha that takes a string and display it
repeating each alphabetical character as many times as its alphabetical index,
followed by a newline.

'a' becomes 'a', 'b' becomes 'bb', 'e' becomes 'eeeee', etc...

Case remains unchanged.

If the number of arguments is not 1, just display a newline.

Examples:

```
$>./repeat_alpha "abc"
abbccc
$>./repeat_alpha "Alex." | cat -e
Allllllllllllleeeeexxxxxxxxxxxxxxxxxxxxxxxx.$
$>./repeat_alpha 'abacadaba 42!' | cat -e
```

```
abbacccaddddabba 42!$
$>./repeat_alpha | cat -e
$
$>
$>./repeat_alpha "" | cat -e
$
$>
==========================================================
================================
```

```c
#include <unistd.h>
int  main(int ac, char **av)
{
     int  letter;
     int  repeat;

     if (ac == 2)
     {
          letter = 0;
          while (av[1][letter])
          {
               repeat = 1;
               if (av[1][letter] >= 'a' && av[1][letter] <= 'z')
                    repeat += av[1][letter] - 'a';
               else if (av[1][letter] >= 'A' && av[1][letter] <=
'Z')
                    repeat += av[1][letter] - 'A';
               while (repeat)
               {
                    write(1, &av[1][letter], 1);
                    repeat--;
               }
               letter++;
          }
     }
     write(1, "\n", 1);
     return (0);
}
===
#include <unistd.h>

int       main(int argc, char *argv[])
{
     int             i;
     int             letter;
     i = 0;
     if (argc == 2)
     {
          while (argv[1][i])
          {
               letter = 0;
               if (argv[1][i] >= 'A' && 'Z' >= argv[1][i])
               {
                    letter = argv[1][i] - 63;
                    while (--letter)
                         write(1, &argv[1][i], 1);
               }
               else if (argv[1][i] >= 'a' && 'z' >= argv[1][i])
               {
                    letter = argv[1][i] - 95;
                    while (--letter)
                         write(1, &argv[1][i], 1);
               }
               else
                    write(1, &argv[1][i], 1);
               i += 1;
          }
     }
     write(1, "\n", 1);
     return (0);
}

===
#include <unistd.h>

int  letter_count(char c)
{
     int  repeat;

     if (c > 'A' && c <= 'Z')
          repeat = c - 'A' + 1;
     else if (c >= 'a' && c <= 'z')
          repeat = c - 'a' + 1;
     else
          repeat = 1;
     return (repeat);
}

int  main(int ac, char **av)
{
     int  repeat;
```

```
    if (ac == 2)
    {
        while (*av[1])
        {
            repeat = letter_count(*av[1]);
                while (repeat--)
                    write(1, av[1], 1);
            av[1]++;
        }
    }
    write(1, "\n", 1);
}

===


========================================./1-0-
search_and_replace.txt====================================
=====
Assignment name  : search_and_replace
Expected files   : search_and_replace.c
Allowed functions: write, exit
-----------------------------------------------------------
----------------------

Write a program called search_and_replace that takes 3
arguments, the first
arguments is a string in which to replace a letter (2nd
argument) by
another one (3rd argument).

If the number of arguments is not 3, just display a
newline.

If the second argument is not contained in the first one
(the string)
then the program simply rewrites the string followed by a
newline.

Examples:
$>./search_and_replace "Papache est un sabre" "a" "o"
Popoche est un sobre
$>./search_and_replace "zaz" "art" "zul" | cat -e
$
```

```
$>./search_and_replace "zaz" "r" "u" | cat -e
zaz$
$>./search_and_replace "jacob" "a" "b" "c" "e" | cat -e
$
$>./search_and_replace "ZoZ eT Dovid oiME le METol." "o"
"a" | cat -e
ZaZ eT David aiME le METal.$
$>./search_and_replace "wNcOre Un ExEmPle Pas Facilw a
Ecrirw " "w" "e" | cat -e
eNcOre Un ExEmPle Pas Facile a Ecrire $
==========================================================
================================
```

```c
#include <unistd.h>

int  main(int argc, char *argv[])
{
    int  i;

    i = 0;
    if (argc == 4)
    {
        if (!argv[2][1] && !argv[3][1])
        {
            while (argv[1][i])
            {
                if (argv[1][i] == argv[2][0])
                    write(1, &argv[3][0], 1);
                else
                    write(1, &argv[1][i], 1);
                i += 1;
            }
        }
    }
    write(1, "\n", 1);
    return (0);
}
```

==

```
======================================./1-0-
ulstr.txt=========================================
Assignment name  : ulstr
Expected files   : ulstr.c
```

Allowed functions: write

```
----------------------------------------------------------
----------------------
```

Write a program that takes a string and reverses the case of all its letters.
Other characters remain unchanged.

You must display the result followed by a '\n'.

If the number of arguments is not 1, the program displays '\n'.

Examples :

```
$>./ulstr "L'eSPrit nE peUt plUs pRogResSer s'Il staGne et
sI peRsIsTent VAnIte et auto-justification." | cat -e
l'EspRIT Ne PEuT PLuS PrOGrESsER S'iL STAgNE ET Si
PErSiStENT vaNiTE ET AUTO-JUSTIFICATION.$
$>./ulstr "S'enTOuRer dE sECreT eSt uN sIGnE De mAnQuE De
coNNaiSSanCe.  " | cat -e
s'ENtoUrER De SecREt EsT Un SigNe dE MaNqUe dE
COnnAIssANcE.  $
$>./ulstr "3:21 Ba  tOut  moUn ki Ka di KE m'en Ka fe fot"
| cat -e
3:21 bA  ToUT  MOuN KI kA DI ke M'EN kA FE FOT$
$>./ulstr | cat -e
$
==========================================================
================================
```

```c
#include <unistd.h>

int  main(int argc, char *argv[])
{
     int       i;
     char      letter;

     i = 0;
     if (argc == 2)
     {
          while (argv[1][i])
          {
               letter = argv[1][i];
               if (argv[1][i] >= 'A' && 'Z' >= argv[1][i])
                    letter += 32;
               if (argv[1][i] >= 'a' && 'z' >= argv[1][i])
                    letter -= 32;
               write(1, &letter, 1);
               i += 1;
          }
     }
     write(1, "\n", 1);
     return (0);
}
===
#include <unistd.h>

void ulstr(char *s)
{
     char c;

     while (*s)
     {
          if (*s >= 'a' && *s <= 'z')
               c = *s - 32;
          else if (*s >= 'A' && *s <= 'Z')
               c = *s + 32;
          else
               c = *s;
          write(1, &c, 1);
          s++;
     }
}

int       main(int argc, char **argv)
{
     if (argc == 2)
          ulstr(argv[1]);
     write(1, "\n", 1);
     return (0);
}
```

```
===================================./1-1-
rot_13.txt========================================
Assignment name  : rot_13
Expected files   : rot_13.c
Allowed functions: write
--------------------------------------------------------------------------------
```

Write a program that takes a string and displays it, replacing each of its letters by the letter 13 spaces ahead in alphabetical order.

'z' becomes 'm' and 'Z' becomes 'M'. Case remains unaffected.

The output will be followed by a newline.

If the number of arguments is not 1, the program displays a newline.

Example:

```
$>./rot_13 "abc"
nop
$>./rot_13 "My horse is Amazing." | cat -e
Zl ubefr vf Nznmvat.$
$>./rot_13 "AkjhZ zLKIJz , 23y " | cat -e
NxwuM mYXVWm , 23l $
$>./rot_13 | cat -e
$
$>
$>./rot_13 "" | cat -e
$
$>
```

===
================================

```
#include <unistd.h>

int       ft_putchar(char c)
{
     return (write(1, &c , 1));
}

void ft_rot_13(char *str)
{
     while (*str++)
     {
          if ('a' <= *(str - 1) && *(str - 1) <= 'z')
```

```
               ft_putchar(((*(str - 1) - 'a' + 13) % 26) + 'a');
          else if ('A' <= *(str - 1) && *(str - 1) <= 'Z')
               ft_putchar(((*(str - 1) - 'A' + 13) % 26) + 'A');
          else
               ft_putchar(*(str - 1));
     }
}

int  main(int ac, char *av[])
{
     if (ac == 2)
          ft_rot_13(av[1]);
     ft_putchar('\n');
     return (0);
}
===
#include <unistd.h>

int  main(int argc, char *argv[])
{
     int  i;
     char mvup;
     char mvdwn;

     i = 0;
     if (argc == 2)
     {
          while (argv[1][i])
          {
               mvup = argv[1][i] + 13;
               mvdwn = argv[1][i] - 13;
               if (('A' <= argv[1][i] && 'M' >= argv[1][i])
                    || ('a' <= argv[1][i] && 'm' >= argv[1][i]))
                         write(1, &mvup, 1);
               else if (('N' <= argv[1][i] && 'Z' >= argv[1][i])
                         || ('n' <= argv[1][i] && 'z' >= argv[1]
[i]))
                    write(1, &mvdwn, 1);
               else
                    write(1, &argv[1][i], 1);
               i += 1;
          }
     }
     write(1, "\n", 1);
     return (0);
}
```

```
===
#include <unistd.h>

int  main(int ac, char **av)
{
    int  i;

    if (ac > 1)
    {
        i = 0;
        while(av[1][i])
        {
            if (av[1][i] >= 'a' && av[1][i] <= 'z')
                av[1][i] = (av[1][i] - 'a' + 13) % 26 + 'a';
            else if (av[1][i] >= 'A' && av[1][i] <= 'Z')
                    av[1][i] = (av[1][i] - 'A' + 13) % 26 +
'A';
            write(1, &av[1][i], 1);
            i++;
        }
    }
    write(1, "\n", 1);
    return (0);
}
```

==================================./1-2-
first_word.txt==================================

Assignment name : first_word
Expected files   : first_word.c
Allowed functions: write

Write a program that takes a string and displays its first word, followed by a newline.

A word is a section of string delimited by spaces/tabs or by the start/end of the string.

If the number of parameters is not 1, or if there are no words, simply display a newline.

Examples:

```
$> ./first_word "FOR PONY" | cat -e
FOR$
$> ./first_word "this        ...       is sparta, then
again, maybe    not" | cat -e
this$
$> ./first_word "   " | cat -e
$
$> ./first_word "a" "b" | cat -e
$
$> ./first_word "  lorem,ipsum  " | cat -e
lorem,ipsum$
$>
==========================================================
================================
```

```
#include <unistd.h>

int  main(int ac, char **av)
{
     if (ac == 2)
     {
          while (*av[1] && (*av[1] == ' ' || *av[1] == '\t' ||
*av[1] == '\n'
                         || *av[1] == '\r' || *av[1] == '\v' ||
*av[1] == '\f'))
               ++av[1];
          while (*av[1] != '\0' && (*av[1] != ' ' && *av[1] !=
'\t' && *av[1] != '\n'
                         && *av[1] != '\r' && *av[1] != '\v' &&
*av[1] != '\f'))
               write(1, av[1]++, 1);
     }
     write(1, "\n", 1);
     return (0);
}
===
#include <unistd.h>
int  ft_isspace(int i)
{
     if (i == '\t' || i == '\n' || i == '\r' || i == '\v' || i
== '\f' || i == ' ')
          return (1);
```

```
    return (0);
}

int  main(int argc, char *argv[])
{
    int  i;

    i = 0;
    if (argc == 2)
    {
        while (ft_isspace((argv[1][i])))
            i += 1;
        while (!(ft_isspace(argv[1][i])) && argv[1][i])
            write(1, &argv[1][i++], 1);
    }
    write(1, "\n", 1);
    return (0);
}
```

```
=======================================./1-2-
ft_putstr.txt==========================================
Assignment name  : ft_putstr
Expected files   : ft_putstr.c
Allowed functions: write
------------------------------------------------------------
----------------------

Write a function that displays a string on the standard
output.

The pointer passed to the function contains the address of
the string's first
character.

Your function must be declared as follows:

void ft_putstr(char *str);
============================================================
=================================
======================================./1-2-
ft_swap.txt=========================================
Assignment name  : ft_swap
Expected files   : ft_swap.c
Allowed functions:
----------------------------------------------------------
----------------------

Write a function that swaps the contents of two integers
the adresses of which
are passed as parameters.

Your function must be declared as follows:

void ft_swap(int *a, int *b);
==========================================================
================================
void ft_swap(int *a, int *b)
{
     int tmp;

     tmp = *a;
     *a = *b;
     *b = tmp;
}

#include <stdio.h>
void ft_swap(int *a, int *b);

int main(void)
{
     int old_age = 223;
     int young_age = 1;
     printf("old_age: %d\nyoung_age: %d\n", old_age, young_age);
     ft_swap(&old_age, &young_age);
     printf("swap them ages!\n");
     printf("old_age: %d\nyoung_age: %d\n", old_age, young_age);
     return (0);
}



======================================./1-3-
first_word.txt=========================================
Assignment name  : first_word
Expected files   : first_word.c
Allowed functions: write
------------------------------------------------------------
----------------------
```

Write a program that takes a string and displays its first word, followed by a newline.

A word is a section of string delimited by spaces/tabs or by the start/end of the string.

If the number of parameters is not 1, or if there are no words, simply display a newline.

Examples:

```
$> ./first_word "FOR PONY" | cat -e
FOR$
$> ./first_word "this        ...       is sparta, then
again, maybe    not" | cat -e
this$
$> ./first_word "   " | cat -e
$
$> ./first_word "a" "b" | cat -e
$
$> ./first_word "  lorem,ipsum  " | cat -e
lorem,ipsum$
$>
============================================================
=================================
```

```
=======================================./1-3-
rev_print.txt==========================================
Assignment name  : rev_print
Expected files   : rev_print.c
Allowed functions: write
------------------------------------------------------------
```

```
----------------------
```

Write a program that takes a string, and displays the
string in reverse
followed by a newline.

If the number of parameters is not 1, the program displays a newline.

Examples:

```
$> ./rev_print "zaz" | cat -e
zaz$
$> ./rev_print "dub0 a POIL" | cat -e
LIOP a 0bud$
$> ./rev_print | cat -e
$
==========================================================
================================
```

```c
#include <unistd.h>

int  main(int ac, char *av[])
{
     int  i;

     if (ac == 2)
     {
          i = 0;
          while (av[1][i])
               i += 1;
          while (i)
               write(1, &av[1][--i], 1);
     }
     write(1, "\n", 1);
     return (0);
}
```

==

```c
#include <unistd.h>

void ft_putchar(char c)
{
     write(1, &c, 1);
}
```

```
int       ft_strlen(char *s)
{
     int  i;

     i = 0;
     while (s[i])
          i++;
     return (i);
}

int       main(int ac, char **av)
{
     int  len;

     if (ac == 2)
     {
          len = ft_strlen(av[1]);
          while (len--)
               write(1, &av[1][len], 1);
     }
     ft_putchar('\n');
}
```

```
=====================================./1-4-
rotone.txt=====================================
Assignment name  : rotone
Expected files   : rotone.c
Allowed functions: write
--------------------------------------------------------------
-----------------------

Write a program that takes a string and displays it,
replacing each of its
letters by the next one in alphabetical order.

'z' becomes 'a' and 'Z' becomes 'A'. Case remains
unaffected.

The output will be followed by a \n.

If the number of arguments is not 1, the program displays
\n.
```

Example:

```
$>./rotone "abc"
bcd
$>./rotone "Les stagiaires du staff ne sentent pas
toujours tres bon." | cat -e
Mft tubhjbjsft ev tubgg of tfoufou qbt upvkpvst usft cpo.$
$>./rotone "AkjhZ zLKIJz , 23y " | cat -e
BlkiA aMLJKa , 23z $
$>./rotone | cat -e
$
$>
$>./rotone "" | cat -e
$
$>
============================================================
=================================
```

```c
#include <unistd.h>

void ft_putchar(char c)
{
    write(1, &c, 1);
}

void rotone(char *s)
{
    while (*s)
    {
        if ((*s >= 'A' && *s <= 'Y') || (*s >= 'a' && *s <=
'y'))
            ft_putchar(*s + 1);
        else if (*s == 'Z' || *s == 'z')
            ft_putchar(*s - 25);
        else
            ft_putchar(*s);
        ++s;
    }
}

int     main(int ac, char **av)
{
    if (ac == 2)
        rotone(av[1]);
    ft_putchar('\n');
```

```
	return (0);
}
===
#include <unistd.h>

int		main(int argc, char *argv[])
{
	int		i;
	char ltr;

	i = 0;
	if (argc == 2)
	{
		while (argv[1][i])
		{
			ltr = argv[1][i];
			if (argv[1][i] >= 'A' && argv[1][i] <= 'Y')
				ltr += 1;
			if (argv[1][i] >= 'a' && argv[1][i] <= 'y')
				ltr += 1;
			if (argv[1][i] == 'Z' || argv[1][i] == 'z')
				ltr -= 25;
			write(1, &ltr, 1);
			i += 1;
		}
	}
	write(1, "\n", 1);
	return (0);
}
```

```
======================================./2-0-
ft_atoi.txt=========================================
Assignment name  : ft_atoi
Expected files   : ft_atoi.c
Allowed functions: None
----------------------------------------------------------
----------------------
```

Write a function that converts the string argument str to
an integer (type int)
and returns it.

It works much like the standard atoi(const char *str) function, see the man.

Your function must be declared as follows:

```
int  ft_atoi(const char *str);
================================================================
===================================
int  ft_atoi(char *str)
{
     int result;
     int sign;

     result = 0;
     sign = 1;
     while (*str == ' ' || (*str >= 9 && *str <= 13))
          str++;
     if (*str == '-')
          sign = -1;
     if (*str == '-' || *str == '+')
          str++;
     while (*str >= '0' && *str <= '9')
     {
          result = result * 10 + *str - '0';
          str++;
     }
     return (sign * result);
}
==
#include <stdio.h>
#include <stdlib.h>

int             ft_atoi(char *str);

int main()
{
     printf("ft_atoi: %d\n", ft_atoi("123456"));
     printf("atoi: %d\n", atoi("123456"));
     printf("ft_atoi: %d\n", ft_atoi("12Three45678"));
     printf("atoi: %d\n", atoi("12Three45678"));
     printf("ft_atoi: %d\n", ft_atoi("Hello World!"));
     printf("atoi: %d\n", atoi("Hello World!"));
     printf("ft_atoi: %d\n", ft_atoi("+42 BLAH!"));
     printf("atoi: %d\n", atoi("+42 BLAH!"));
     printf("ft_atoi: %d\n", ft_atoi("-42"));
     printf("atoi: %d\n", atoi("-42"));
     printf("ft_atoi: %d\n", ft_atoi("     +42"));
```

```
    printf("atoi: %d\n", atoi("     +42"));
    printf("ft_atoi: %d\n", ft_atoi("\t\n\v\f\r 42"));
    printf("atoi: %d\n", atoi("\t\n\v\f\r 42"));
    printf("ft_atoi: %d\n", ft_atoi("5"));
    printf("atoi: %d\n", atoi("5"));

    return 0;
}
```

```
======================================./2-0-
ft_strdup.txt=========================================
Assignment name  : ft_strdup
Expected files   : ft_strdup.c
Allowed functions: malloc
----------------------------------------------------------
----------------------

Reproduce the behavior of the function strdup (man
strdup).

Your function must be declared as follows:

char    *ft_strdup(char *src);
==========================================================
================================
```

```
#include <stdlib.h>

char *ft_strdup(char *src)
{
    char *dup;
    char *sptr;
    char *dptr;

    sptr = src;
    while (*sptr++)
        ;
    dup = malloc(sptr - src + 1);
    if (!dup)
        return (NULL);
    dptr = dup;
    while ((*dptr++ = *src++) != '\0')
        ;
    return (dup);
}

===
#include <stdlib.h>

char *ft_strdup(char *src)
{
    int       i;
    int       length;
    char *strcpy;

    length = 0;
    while (src[length])
        length++;
    strcpy = malloc(length + 1);
    if (strcpy != NULL)
    {
        i = 0;
        while (src[i])
        {
            strcpy[i] = src[i];
            i++;
        }
        strcpy[i] = '\0';
    }
    return (strcpy);
}

===
#include <stdio.h>
char    *ft_strdup(char *src);

int main(void) {
    char *greet = "Salut";
    char *test1 = "Gonna pass this test, even if I gotta dup!
\n";
    char *test2 = ft_strdup(test1);

    printf("%s\n", ft_strdup(greet));
    printf("test1: %s", test1);
    printf("test2: %s", test2);
    return 0;
}
```

======================================./2-0-inter.txt===

Assignment name : inter
Expected files   : inter.c
Allowed functions: write

Write a program that takes two strings and displays, without doubles, the characters that appear in both strings, in the order they appear in the first one.

The display will be followed by a \n.

If the number of arguments is not 2, the program displays \n.

Examples:

```
$>./inter "padinton" "paqefwtdjetyiytjneytjoeyjnejeyj" | cat -e
padinto$
$>./inter ddf6vewg64f gtwthgdwthdwfteewhrtag6h4ffdhsd | cat -e
df6ewg4$
$>./inter "rien" "cette phrase ne cache rien" | cat -e
rien$
$>./inter | cat -e
$
```

==

```
#include <unistd.h>

int  iter(char *str, char c, int len)
{
     int  i;

     i = 0;
     while (str[i] && (i < len || len == -1))
```

```
          if (str[i++] == c)
               return (1);
     return (0);
}

int  main(int argc, char *argv[])
{
     int  i;

     if (argc == 3)
     {
          i = 0;
          while (argv[1][i])
          {
               if (!iter(argv[1], argv[1][i], i) &&
iter(argv[2], argv[1][i], -1))
                    write(1, &argv[1][i], 1);
               i += 1;
          }
     }
     write(1, "\n", 1);
     return (0);
}
===
#include <unistd.h>

int  scan(char *str, char c, int nb)
{
     while (nb >= 0)
     {
          if (str[nb] == c)
               return (0);
          nb--;
     }
     return (1);
}

void inter(char *str1, char *str2)
{
     int i = 0;
     int  j;

     while(str1[i])
     {
          j = 0;
          while(str2[j])
          {
                if (str1[i] == str2[j])
                {
                    if (scan(str1, str1[i], i - 1))
                        write(1, &str1[i], 1);
                    break;
                }
                j++;
            }
            i++;
        }
}

int main(int argc, char **argv)
{
    if (argc == 3)
        inter(argv[1], argv[2]);
    write(1, "\n", 1);
    return (0);
}
```

==./2-0-last_word.txt==

Assignment name : last_word
Expected files   : last_word.c
Allowed functions: write

Write a program that takes a string and displays its last word followed by a \n.

A word is a section of string delimited by spaces/tabs or by the start/end of
the string.

If the number of parameters is not 1, or there are no words, display a newline.

Example:

```
$> ./last_word "FOR PONY" | cat -e
PONY$
$> ./last_word "this        ...       is sparta, then
```

```
again, maybe    not" | cat -e
not$
$> ./last_word "   " | cat -e
$
$> ./last_word "a" "b" | cat -e
$
$> ./last_word "  lorem,ipsum  " | cat -e
lorem,ipsum$
$>
==========================================================
================================
#include <unistd.h>

void last_word(char *str)
{
     int  j;
     int i;

     i = 0;
     j = 0;
     while (str[i])
     {
          if (str[i] == ' ' && str[i + 1] >= 33 && str[i + 1] <=
126)
               j = i + 1;
          i++;
     }
     while (str[j] >= 33 && str[j] <= 127)
     {
          write(1, &str[j], 1);
          j++;
     }
}

int       main(int argc, char **argv)
{
     if (argc == 2)
          last_word(argv[1]);
     write(1, "\n", 1);
     return (0);
}
```

```
======================================./2-0-
```

reverse_bits.txt==
Assignment name : reverse_bits
Expected files   : reverse_bits.c
Allowed functions:

Write a function that takes a byte, reverses it, bit by bit (like the example) and returns the result.

Your function must be declared as follows:

```
unsigned char reverse_bits(unsigned char octet);
```

Example:

```
  1 byte
_____________
 0010  0110
     ||
     \/
 0110  0100
```

===

```
unsigned char  reverse_bits(unsigned char octet)
{
	unsigned char  res = 0;
	int i = 8;

	while (i > 0)
	{
		res = res * 2 + (octet % 2);
		octet = octet / 2;
		i--;
	}
	return (res);
}
```

===

```
unsigned char  reverse_bits(unsigned char octet)
{
	return (((octet >> 0) & 1) << 7) | \
	        (((octet >> 1) & 1) << 6) | \
	        (((octet >> 2) & 1) << 5) | \
	        (((octet >> 3) & 1) << 4) | \
             (((octet >> 4) & 1) << 3) | \
             (((octet >> 5) & 1) << 2) | \
             (((octet >> 6) & 1) << 1) | \
             (((octet >> 7) & 1) << 0);
}
===
#include <stdio.h>

unsigned char   reverse_bits(unsigned char octet);

int     main()
{
     printf("%d", reverse_bits(38));
     return (0);
}

/* ******************************************
** echo 00100110 | perl -lpe '$_=pack"B*",$_'
** echo "d" | perl -lpe '$_=unpack"B*"'
** ;; Convert binary to ascii with perl;
**
** echo "&" | perl -lpe '$_=unpack"B*"'
** echo 01100100 | perl -lpe '$_=pack"B*",$_'
** ;; Use perl to convert ascii char to binary
**
** echo "&" | perl -lpe '$_=unpack"B*"' && echo "d" | perl -lpe
'$_=unpack"B*"'
** ;; See the bits reversed more easily
** ******************************************
*/
```

```
======================================./2-0-
swap_bits.txt=========================================
Assignment name  : swap_bits
Expected files   : swap_bits.c
Allowed functions:
----------------------------------------------------------
----------------------

Write a function that takes a byte, swaps its halves (like
the example) and
returns the result.
```

Your function must be declared as follows:

```
unsigned char swap_bits(unsigned char octet);
```

Example:

```
  1 byte
_____________
 0100 | 0001
     \ /
     / \
 0001 | 0100
=================================================================
====================================
```

```
unsigned char  swap_bits(unsigned char c)
{
     return ((c >> 4) | (c << 4));
}
===
#include <unistd.h>
#include <stdio.h>
#include <ctype.h>

unsigned char   swap_bits(unsigned char c);
int asciiToBinary(int input);

int  main(void)
{
     char c;

     c = 't';
     write(1, &c, 1);
     write(1, "\n", 1);
     printf("%08d %c\n", asciiToBinary(toascii(c)), c);
     c = swap_bits(c);
     printf("%08d %c\n", asciiToBinary(toascii(c)), c);
     write(1, &c, 1);
     write(1, "\n", 1);

     return (0);
}


int asciiToBinary(int input)
{
    int result = 0, i = 1, remainder;

    /* convert decimal to binary format */
    while (input > 0) {
        remainder = input % 2;
        result = result + (i * remainder);
        input = input / 2;
        i = i * 10;
    }

    /* print the resultant binary value */
    return(result);
}

==
#include <stdio.h>

unsigned char swap_bits(unsigned char c);

int main(void)
{
    char letter_t;
    char letter_G;
    letter_t = 't';
    letter_G = 'G';

    printf("letter_t after swap: %c", swap_bits(letter_t));
    printf("\n");
    printf("letter_G after swap: %c", swap_bits(letter_G));
    printf("\n");
    return (0);
}
```

```
======================================./2-0-
union.txt=========================================
Assignment name  : union
Expected files   : union.c
Allowed functions: write
----------------------------------------------------------
----------------------
```

Write a program that takes two strings and displays,
without doubles, the
characters that appear in either one of the strings.

The display will be in the order characters appear in the command line, and will be followed by a \n.

If the number of arguments is not 2, the program displays \n.

Example:

```
$>./union zpadinton "paqefwtdjetyiytjneytjoeyjnejeyj" |
cat -e
zpadintoqefwjy$
$>./union ddf6vewg64f gtwthgdwthdwfteewhrtag6h4ffdhsd |
cat -e
df6vewg4thras$
$>./union "rien" "cette phrase ne cache rien" | cat -e
rienct phas$
$>./union | cat -e
$
$>
$>./union "rien" | cat -e
$
$>
===========================================================
================================
```

```c
#include <unistd.h>

int       not_seen_before(char *s, int max_pos, char c)
{
     int  i;

     i = -1;
     while(++i < max_pos)
          if (s[i] == c)
               return (0);
     return (1);
}

void ft_union(char *s1, char *s2)
{
     int  i;
     int  j;

     i = -1;
```

```
	while (s1[++i])
		if (not_seen_before(s1, i, s1[i]))
			write(1, &s1[i], 1);
	j = -1;
	while (s2[++j])
		if (not_seen_before(s1, i, s2[j]) &
not_seen_before(s2, j, s2[j]))
			write(1, &s2[j], 1);
}

int  main(int ac, char **av)
{
	if (ac == 3)
		ft_union(av[1], av[2]);
	write(1, "\n", 1);
	return (0);
}
```

```
=========================================./2-1-
alpha_mirror.txt=========================================
Assignment name  : alpha_mirror
Expected files   : alpha_mirror.c
Allowed functions: write
----------------------------------------------------------------------------
-------------------
```

Write a program called alpha_mirror that takes a string and displays this string after replacing each alphabetical character by the opposite alphabetical character, followed by a newline.

'a' becomes 'z', 'Z' becomes 'A'
'd' becomes 'w', 'M' becomes 'N'

and so on.

Case is not changed.

If the number of arguments is not 1, display only a newline.

Examples:

```
$>./alpha_mirror "abc"
zyx
$>./alpha_mirror "My horse is Amazing." | cat -e
Nb slihv rh Znzarmt.$
$>./alpha_mirror | cat -e
$
$>
==========================================================
================================
```

```
#include <unistd.h>

int  main(int argc, char *argv[])
{
     int       i;
     char ltr;

     i = 0;
     if (argc == 2)
     {
          while (argv[1][i])
          {
               ltr = argv[1][i];
               if ('A' <= argv [1][i] && 'Z' >= argv[1][i])
                    ltr = 'Z' - argv[1][i] + 'A';
               if ('a' <= argv[1][i] && 'z' >= argv[1][i])
                    ltr = 'z' -argv[1][i] + 'a';
               write(1, &ltr, 1);
               i += 1;
          }
     }
     write(1, "\n", 1);
     return (0);
}
===
#include <unistd.h>

void ft_putchar(char c)
{
     write(1, &c, 1);
}

void alpha_mirror(char *str)
{
     int i;
```

```
    i = 0;
    while(str[i])
    {
        if (str[i] >= 'a' && str[i] <= 'z')
            ft_putchar('z' - (str[i] - 'a'));
        else if (str[i] >= 'A' && str[i] <= 'Z')
                ft_putchar('Z' - (str[i] - 'A'));
        else
            ft_putchar(str[i]);
        i++;
    }
}

int main(int argc, char **argv)
{
    if (argc == 2)
        alpha_mirror(argv[1]);
    write(1, "\n", 1);
    return (0);
}
```

```
=====================================./2-1-
max.txt======================================
Assignment name  : max
Expected files   : max.c
Allowed functions:
----------------------------------------------------------
---------------------

Write the following function:

int      max(int* tab, unsigned int len);

The first parameter is an array of int, the second is the
number of elements in
the array.

The function returns the largest number found in the
array.

If the array is empty, the function returns 0.
==========================================================
===============================
```

```c
#include <stdio.h>

int max(int *tab, unsigned int len)
{
     if (len == 0)
          return (0);
     int highest = -2147482648;
     unsigned int i = 0;
     while (i < len)
     {
          if (tab[i] > highest)
               highest = tab[i];
          i++;
     }
     return (highest);
}
```

==

```c
int max(int *tab, unsigned int len)
{
     int max;

     if (!len)
          return (0);
     max = tab[--len];
     while (len--)
          if (tab[len] > max)
               max = tab[len];
     return (max);
}
```

==

```c
#include <stdio.h>

int max(int *tab, unsigned int len);

int  main(void)
{
     int nums01[] = {-2, -3, -776, -9};
     printf("%d\n", max(nums01, 4));
     int nums02[] = {-2, 101, 23};
     printf("%d\n", max(nums02, 3));
     int nums03[] = {-2, 101, 23, 200, -2000, 4000, 3999, 89};
     printf("%d\n", max(nums03, 8));
     return (0);
}
```

```
======================================./2-3-
wdmatch.txt======================================
Assignment name  : wdmatch
Expected files   : wdmatch.c
Allowed functions: write
--------------------------------------------------------------
----------------------

Write a program that takes two strings and checks whether
it's possible to
write the first string with characters from the second
string, while respecting
the order in which these characters appear in the second
string.

If it's possible, the program displays the string,
followed by a \n, otherwise
it simply displays a \n.

If the number of arguments is not 2, the program displays
a \n.

Examples:

$>./wdmatch "faya" "fgvvfdxcacpolhyghbreda" | cat -e
faya$
$>./wdmatch "faya" "fgvvfdxcacpolhyghbred" | cat -e
$
$>./wdmatch "quarante deux" "qfqfsudf arzgsayns tsregfdgs
sjytdekuoixq " | cat -e
quarante deux$
$>./wdmatch "error" rrerrrfiiljdfxjyuifrrvcoojh | cat -e
$
$>./wdmatch | cat -e
$
==============================================================
================================
```

```c
#include <unistd.h>

void wdmatch(char *s1, char *s2)
{
    int len;
    int i;

    len = 0;
    i = 0;
    while (s1[len])
        ++len;
    while (*s2 && i < len)
        (*s2++ == s1[i]) ? ++i : 0;
    if (i == len)
        write(1, s1, len);
}

int main(int ac, char **av)
{
    if (ac == 3)
        wdmatch(av[1], av[2]);
    write(1, "\n", 1);
    return (0);
}
==
#include <unistd.h>

void ft_putstr(char const *str)
{
    int i;

    i = 0;
    while (str[i])
        write(1, &str[i++], 1);
}

int  main(int argc, char const *argv[])
{
    int i;
    int j;

    if (argc == 3)
    {
        i = 0;
        j = 0;
        while (argv[2][j])
            if (argv[2][j++] == argv[1][i])
                i += 1;
        if (!argv[1][i])
            ft_putstr(argv[1]);
    }
```

```
     write(1, "\n", 1);
     return (0);
}
==
#include <unistd.h>

int ft_strlen(char *str)
{
     int i;

     i = 0;
     while (str[i])
          i++;
     return (i);
}

void ft_putstr(char *str)
{
     int i;

     i = 0;
     while (str[i])
     {
          write(1, &str[i], 1);
          i++;
     }
}

void wdmatch(char *s1, char *s2)
{
     int i;
     int j;

     i = 0;
     j = 0;
     while (s2[i] && s1[j])
     {
          if (s2[i] == s1[j])
               j++;
          i++;
     }
     if (j == ft_strlen(s1))
          ft_putstr(s1);
}

int main(int argc, char **argv)
{
     if (argc == 3)
          wdmatch(argv[1], argv[2]);
     write(1, "\n", 1);
     return (0);
}
```

=====================================./2-4-do_op.txt===

```
Assignment name  : do_op
Expected files   : *.c, *.h
Allowed functions: atoi, printf, write
---------------------------------------------------------------------------------
```

Write a program that takes three strings:
- The first and the third one are representations of base-10 signed integers
  that fit in an int.
- The second one is an arithmetic operator chosen from: + - * / %

The program must display the result of the requested arithmetic operation,
followed by a newline. If the number of parameters is not 3, the program
just displays a newline.

You can assume the string have no mistakes or extraneous characters. Negative
numbers, in input or output, will have one and only one leading '-'. The
result of the operation fits in an int.

Examples:

```
$> ./do_op "123" "*" 456 | cat -e
56088$
$> ./do_op "9828" "/" 234 | cat -e
42$
$> ./do_op "1" "+" "-43" | cat -e
-42$
$> ./do_op | cat -e
```

```
$
==========================================================
================================
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
     if (argc == 4)
     {
          if (argv[2][0] == '+')
               printf("%d", (atoi(argv[1]) + atoi(argv[3])));
          if (argv[2][0] == '-')
               printf("%d", (atoi(argv[1]) - atoi(argv[3])));
          if (argv[2][0] == '*')
               printf("%d", (atoi(argv[1]) * atoi(argv[3])));
          if (argv[2][0] == '/')
               printf("%d", (atoi(argv[1]) / atoi(argv[3])));
          if (argv[2][0] == '%')
               printf("%d", (atoi(argv[1]) % atoi(argv[3])));
     }
     printf("\n");
     return (0);
}
```

```
======================================./2-4-
print_bits.txt=========================================
Assignment name  : print_bits
Expected files   : print_bits.c
Allowed functions: write
----------------------------------------------------------
----------------------

Write a function that takes a byte, and prints it in
binary WITHOUT A NEWLINE
AT THE END.

Your function must be declared as follows:

void print_bits(unsigned char octet);

Example, if you pass 2 to print_bits, it will print
```

```
"00000010"
===========================================================
================================
#include <unistd.h>

void print_bits(unsigned char octet)
{
	int div = 128;
	int num = octet;

	while (div != 0)
	{
		if (div <= num)
		{
			write(1, "1", 1);
			num = num % div;
		}
		else
			write(1, "0", 1);
		div = div / 2;
	}
}
===
#include <unistd.h>

void print_bits(unsigned char octet)
{
	int  i;
	unsigned char bit;

	i = 8;
	while (i--)
	{
		bit = (octet >> i & 1) + '0';
		write(1, &bit, 1);
	}
}

==
#include <unistd.h>

void print_bits(unsigned char octet)
{
	int i;
	char c;
```

```
    i = 128;
    while (i > 0)
    {
        if (octet < i)
        {
            c = '0';
            i = i / 2;
            write(1, &c, 1);
        }
        else
        {
            c = '1';
            write(1, &c, 1);
            octet = octet - i;
            i = i / 2;
        }
    }
}
==
#include <unistd.h>

void print_bits(unsigned char octet);

int main(void)
{
    print_bits(0);
    write(1, "\n", 1);
    print_bits(1);
    write(1, "\n", 1);
    print_bits(2);
    write(1, "\n", 1);
    print_bits(10);
    write(1, "\n", 1);
    print_bits(113);
    write(1, "\n", 1);
    print_bits(255);
    write(1, "\n", 1);
    return (0);
}
```

```
======================================./2-5-
ft_strcmp.txt=========================================
Assignment name  : ft_strcmp
```

```
Expected files   : ft_strcmp.c
Allowed functions:
----------------------------------------------------------
----------------------

Reproduce the behavior of the function strcmp (man
strcmp).

Your function must be declared as follows:

int    ft_strcmp(char *s1, char *s2);
==========================================================
================================
```

```c
int ft_strcmp(char *s1, char *s2)
{
    while (*s1++ == *s2++)
        if (!*s1 && !*s2)
            return (0);
    return (*--s1 - *--s2);
}
==
int  ft_strcmp(char *s1, char *s2)
{
    while (*s1 && (*s1 == *s2))
    {
        s1 += 1;
        s2 += 1;
    }
    return (*(unsigned char*)s1 - *(unsigned char*)s2);
}
==
#include <stdio.h>

int ft_strcmp(char *s1, char *s2)
{
    int i;

    i = 0;
    while (s1[i])
    {
        if (s1[i] != s2[i])
            return (s1[i] - s2[i]);
        i++;
    }
    return (s1[i] - s2[i]);
}

==
#include <stdio.h>

int ft_strcmp(char *s1, char *s2);

int main(int argc, char **argv)
{
     if (argc == 3)
          printf("%d\n", ft_strcmp(argv[1], argv[2]));
     return (0);
}
```

```
======================================./2-5-
ft_strrev.txt=========================================
Assignment name  : ft_strrev
Expected files   : ft_strrev.c
Allowed functions:
-----------------------------------------------------------
----------------------

Write a function that reverses (in-place) a string.

It must return its parameter.

Your function must be declared as follows:

char    *ft_strrev(char *str);
===========================================================
================================
```

```
char *ft_strrev(char *str)
{
     int i;
     int len;
     char tmp;

     len = 0;
     while (str[len])
          len++;
     i = -1;
     while (++i < --len)
     {
          tmp = str[i];
          str[i] = str[len];
          str[len] = tmp;
     }
     return (str);
}
===
char *ft_strrev(char *str)
{
     char temp;
     int length = 0;
     int i = 0;

     length = 0;
     i = 0;
     while (str[length])
          length++;
     while (i < (length - 1))
     {
          temp = str[i];
          str[i] = str[length - 1];
          str[length - 1] = temp;
          i++;
          length--;}
     return (str);
}
==
void ft_swap(char *a, char *b)
{
     char tmp;

     tmp = *a;
     *a = *b;
     *b = tmp;
}

char *ft_strrev(char *str)
{
     char *begin;
     char *end;

     begin = str;
     end = str;
     while (*end)
          end++;
     end--;
     while (begin < end)
     {
          ft_swap(begin, end);
          begin++;
          end--;
     }
     return (str);
}
==
#include <stdio.h>
#include <unistd.h>

char *ft_strrev(char *str);

int main(int argc, char **argv)
{
     int i;

     i = 0;
     if (argc != 2)
     {
          write(1, "\n", 1);
          return (0);
     }
     else
          printf("%s\n", ft_strrev(argv[1]));
     return (0);
}
```

```
======================================./2-6-is_power_of_2.txt=========================================
Assignment name  : is_power_of_2
Expected files   : is_power_of_2.c
Allowed functions: None
--------------------------------------------------------------------------------
```

Write a function that determines if a given number is a power of 2.

This function returns 1 if the given number is a power of 2, otherwise it returns 0.

Your function must be declared as follows:

```
int      is_power_of_2(unsigned int n);
===============================================================
===================================
int is_power_of_2(unsigned int n)
{
     if (n == 0)
          return (0);
     else
          return ((n & (-n)) == n ? 1: 0);
}
==
int is_power_of_2(unsigned int n)
{
     if (n == 0)
          return (0);
     while (n % 2 == 0)
          n /= 2;
     return ((n == 1) ? 1 : 0);
}

==
int is_power_of_2(unsigned int n)
{
     if (n == 2 || n == 1)
          return (1);
     if (n == 0)
          return (0);
     while (n % 2 == 1)
          return (0);
     while (n > 2)
     {
          if (n % 2 == 1)
               return (0);
          n = n / 2;
     }
     return (1);
}

==
#include <stdio.h>

int is_power_of_2(unsigned int n);

int main(void)
{
	unsigned int num[7];
	num[0] = 0;
	num[1] = 200;
	num[2] = 32;
	num[3] = 256;
	num[4] = 13;
	num[5] = 1000;
	num[6] = 1024;
	int i;

	i = 0;
	while(i <= 6)
	{
	if (is_power_of_2(num[i]))
		printf("%s %d\n", "yep", num[i]);
	else
		printf("%s %d\n", "nope", num[i]);
	i++;
	}
}
```

```
======================================./3-0-
add_prime_sum.txt=========================================
Assignment name  : add_prime_sum
Expected files   : add_prime_sum.c
Allowed functions: write, exit
----------------------------------------------------------
----------------------
```

Write a program that takes a positive integer as argument and displays the sum of all prime numbers inferior or equal to it followed by a newline.

If the number of arguments is not 1, or the argument is not a positive number, just display 0 followed by a newline.

Yes, the examples are right.

Examples:

```
$>./add_prime_sum 5
10
$>./add_prime_sum 7 | cat -e
17$
$>./add_prime_sum | cat -e
0$
$>
===========================================================
================================
```

```
#include <unistd.h>

int ft_atoi(char *str)
{
    int result;
    int sign;

    result = 0;
    sign = 1;
    while (*str == ' ' || (*str >= 9 && *str <= 13))
        str++;
    if (*str == '-')
        sign = -1;
    if (*str == '-' || *str == '+')
        str++;
    while (*str >= '0' && *str <= '9')
    {
        result = result * 10 + *str - '0';
        str++;
    }
    return (sign * result);
}

void ft_putnbr(int nb)
{
    char c;

    if (nb < 0)
    {
        nb = -nb;
        write(1, "-", 1);
    }
    if (nb < 10)
    {
        c = nb + '0';
        write(1, &c, 1);
```

```
     }
     else
     {
          ft_putnbr(nb / 10);
          ft_putnbr(nb % 10);
     }
}

int  is_prime(int nb)
{
     int i;

     i = 2;
     if (nb <= 1)
          return (0);
     while (i <= (nb / 2))
     {
          if (!(nb % i))
               return (0);
          else
               i += 1;
     }
     return (1);
}

/* *****************************
** Another way to write is_prime
int is_prime(int num)
{
     int i;

     i = 3;
     if (num <= 1)
          return (0);
     if (num % 2 == 0 && num > 2)
          return (0);
     while (i < (num / 2))
     {
          if (num % i == 0)
               return 0;
          i += 2;
     }
     return 1;
}
** ****************************
*/
```

```
int  main(int argc, char *argv[])
{
     int  nb;
     int sum;

     if (argc == 2)
     {
          nb = ft_atoi(argv[1]);
          sum = 0;
          while (nb > 0)
               if (is_prime(nb--))
                    sum += (nb + 1);
          ft_putnbr(sum);
     }
     if (argc != 2)
          ft_putnbr(0);
     write(1, "\n", 1);
     return (0);
}
```

======================================./3-0-epur_str.txt===

Assignment name : epur_str
Expected files   : epur_str.c
Allowed functions: write

Write a program that takes a string, and displays this string with exactly one space between words, with no spaces or tabs either at the beginning or the end, followed by a \n.

A "word" is defined as a part of a string delimited either by spaces/tabs, or by the start/end of the string.

If the number of arguments is not 1, or if there are no words to display, the program displays \n.

Example:

```
$> ./epur_str "vous voyez c'est facile d'afficher la meme
chose" | cat -e
vous voyez c'est facile d'afficher la meme chose$
$> ./epur_str " seulement          la c'est      plus dur
" | cat -e
seulement la c'est plus dur$
$> ./epur_str "comme c'est cocasse" "vous avez entendu,
Mathilde ?" | cat -e
$
$> ./epur_str "" | cat -e
$
$>
==========================================================
================================
```

```c
#include <unistd.h>

int   main(int argc, char const *argv[])
{
    int i;
    int flg;

    if (argc == 2)
    {
        i = 0;
        while (argv[1][i] == ' ' || argv[1][i] == '\t')
            i += 1;
        while (argv[1][i])
        {
            if (argv[1][i] == ' ' || argv[1][i] == '\t')
                flg = 1;
            if (!(argv[1][i] == ' ' || argv[1][i] == '\t'))
            {
                if (flg)
                    write(1, " ", 1);
                flg = 0;
                write(1, &argv[1][i], 1);
            }
            i += 1;
        }
    }
    write(1, "\n", 1);
    return (0);
}
===
#include <unistd.h>

void my_putchar(char c)
{
     write(1, &c, 1);
}

void epur_str(char *str)
{
     char sp;
     int i;

     sp = -1;
     i = 0;
     while(str[i])
     {
          if (str[i] != ' ' && str[i] != '\t')
          {
               if (sp == 1)
                    my_putchar(' ');
               sp = 0;
               my_putchar(str[i]);
          }
          else if (sp == 0)
               sp = 1;
          i++;
     }
}

int main(int argc, char **argv)
{
     if (argc == 2)
          epur_str(argv[1]);
     my_putchar('\n');
     return (0);
}
```

```
======================================./3-0-
ft_list_size.txt=========================================
Assignment name  : ft_list_size
Expected files   : ft_list_size.c, ft_list.h
Allowed functions:
----------------------------------------------------------
----------------------
```

Write a function that returns the number of elements in the linked list that's
passed to it.

It must be declared as follows:

```
int  ft_list_size(t_list *begin_list);
```

You must use the following structure, and turn it in as a file called
ft_list.h:

```
typedef struct    s_list
{
    struct s_list *next;
    void          *data;
}                 t_list;
```

===
================================

```
#include "3-0-ft_list.h"

int  ft_list_size(t_list *begin_list)
{
     int i;

     i = 0;
     while (begin_list)
     {
          begin_list = begin_list->next;
          ++i;
     }
     return (i);
}
==
#include <stdlib.h>
#include <stdio.h>
#include "3-0-ft_list.h"

int     ft_list_size(t_list *begin_list);

t_list *new(void *data)
{
     t_list *n;
```

```
	n = (t_list *)malloc(sizeof(t_list));
	if (n)
	{
		n->data = data;
		n->next = NULL;
	}
	return (n);
}

int main(void)
{
	t_list *p, *s, *j, *t;

	p = new("one");
	s = new("two");
	j = new("three");
	t = new("four");
	p->next = s;
	s->next = j;
	j->next = t;
	printf("%d\n", ft_list_size(p));
	return (0);
}
```

======================================./3-0-
ft_rrange.txt===
Assignment name : ft_rrange
Expected files   : ft_rrange.c
Allowed functions: malloc

Write the following function:

```
int     *ft_rrange(int start, int end);
```

It must allocate (with malloc()) an array of integers,
fill it with consecutive
values that begin at end and end at start (Including start
and end !), then
return a pointer to the first value of the array.

Examples:

```
- With (1, 3) you will return an array containing 3, 2 and
1
- With (-1, 2) you will return an array containing 2, 1, 0
and -1.
- With (0, 0) you will return an array containing 0.
- With (0, -3) you will return an array containing -3, -2,
-1 and 0.
============================================================
=================================
```

```
#include <stdlib.h>

int *ft_rrange(int start, int end)
{
    int *r;
    int len;

    len = (end >= start) ? end - start + 1 : start - end + 1;
    if (!(r = (int*) malloc(sizeof(int) * len)))
        return (NULL);
    while (len--)
        r[len] = (end >= start) ? start++ : start--;
    return (r);
}
===
#include <stdlib.h>

int *ft_rrange(int start, int end)
{
    int *rrange;
    int i;

    if (start > end)
        rrange = (int *)malloc(sizeof(int) * (start - end) +
1);
    else
        rrange = (int *)malloc(sizeof(int) * (end - start) +
1);
    i = 0;
    while (start != end)
    {
        rrange[i++] = end;
        end -= (start > end) ? -1 : 1;
    }
    rrange[i] = end;
    return (rrange);
```

```
}

===
#include <stdlib.h>

int ft_abs(int i)
{
     if (i < 0)
          return (-i);
     return (i);
}

int *ft_range(int start, int end)
{
     int *tab;
     int i;

     i = 0;
     while ((start + i) <= end)
          i++;
     if (!(tab = (int *)malloc(sizeof(int) * i)))
          return (NULL);
     i = -1;
     while ((start + ++i) <= end)
          tab[i] = start + i;
     return (tab);
}

int *ft_rangei(int start, int end)
{
     int *tab;
     int i;

     i = 0;
     while((start + i) <= end)
          i++;
     if (!(tab = (int *)malloc(sizeof(int) * i)))
          return (NULL);
     i = -1;
     while ((end - ++i) >= start)
          tab[i] = end - i;
     return (tab);
}

int *ft_rrange(int start, int end)
{
     if (start < end)
```

```
        return (ft_rangei(start, end));
    return (ft_range(end, start));
}

===
#include <stdlib.h>

int *ft_rrange(int start, int end)
{
    int *range;
    int i;
    int n;

    i = 0;
    if (start > end)
        return (ft_rrange(end, start));
    n = end - start + 1;
    range = (int *)malloc(sizeof(int) * n);
    if (range)
    {
        while (i < n)
        {
            range[i] = start;
            start++;
            i++;
        }
    }
    return (range);
}

===
#include <stdio.h>
int *ft_rrange(int start, int end);

int main(void)
{
    int i;
    int *prt;

    i = 0;
    prt = ft_rrange(1, 3);
    while(i <= 2)
    {
        printf("%d ", prt[i]);
        i++;
    }
    printf("\n");

    i = 0;
    prt = ft_rrange(-1, 2);
    while(i <= 3)
    {
        printf("%d ", prt[i]);
        i++;
    }
    printf("\n");

    i = 0;
    prt = ft_rrange(0, 0);
    while(i <= 0)
    {
        printf("%d ", prt[i]);
        i++;
    }
    printf("\n");

    i = 0;
    prt = ft_rrange(0, -3);
    while(i <= 3)
    {
        printf("%d ", prt[i]);
        i++;
    }
    printf("\n");

    return (0);
}
```

```
======================================./3-0-hidenp.txt=========================================
Assignment name  : hidenp
Expected files   : hidenp.c
Allowed functions: write
--------------------------------------------------------------------------------
```

Write a program named hidenp that takes two strings and displays 1 followed by a newline if the first string is hidden in the second one, otherwise displays 0 followed by a newline.

Let s1 and s2 be strings. We say that s1 is hidden in s2 if it's possible to
find each character from s1 in s2, in the same order as they appear in s1.
Also, the empty string is hidden in any string.

If the number of parameters is not 2, the program displays a newline.

Examples :

```
$>./hidenp "fgex.;" "tyf34gdgf;'ektufjhgdgex.;.;rtjynur6" | cat -e
1$
$>./hidenp "abc" "2altrb53c.sse" | cat -e
1$
$>./hidenp "abc" "btarc" | cat -e
0$
$>./hidenp | cat -e
$
$>
============================================================
=================================
```

```c
#include <unistd.h>

int main(int ac, char **av)
{
    int i;
    int j;
    j = 0;
    i = 0;
    if (ac == 3)
    {
        while (av[2][j] && av[1][i])
        {
            if (av[2][j] == av[1][i])
                i++;
            j++;
        }
        if (av[1][i] == '\0')
            write(1, "1", 1);
        else
```

```
               write(1, "0", 1);
     }
     write(1, "\n", 1);
     return (0);
}

==
#include <unistd.h>

void hidenp(char *s1, char *s2)
{
     while (*s2)
     {
          if (*s1 && *s1 == *s2)
               s1++;
          s2++;
     }
     if (!*s1)
          write(1, "1", 1);
     else
          write(1, "0", 1);
}

int main(int argc, char **argv)
{
     if (argc == 3)
          hidenp(argv[1], argv[2]);
     write(1, "\n", 1);
     return (0);
}

==
#include <unistd.h>

void hidenp(char *s1, char *s2)
{
     while (*s2)
          if (*s1 == *s2++)
               s1++;
     (*s1 == '\0') ? write(1, "1", 1) : write(1, "0", 1);
}

int main(int argc, char **argv)
{
     if (argc == 3)
          hidenp(argv[1], argv[2]);
```

```c
    write(1, "\n", 1);
    return (0);
}
```

```
======================================./3-0-
pgcd.txt=========================================
Assignment name  : pgcd
Expected files   : pgcd.c
Allowed functions: printf, atoi, malloc, free
----------------------------------------------------------
----------------------

Write a program that takes two strings representing two
strictly positive
integers that fit in an int.

Display their highest common denominator followed by a
newline (It's always a
strictly positive integer).

If the number of parameters is not 2, display a newline.

Examples:

$> ./pgcd 42 10 | cat -e
2$
$> ./pgcd 42 12 | cat -e
6$
$> ./pgcd 14 77 | cat -e
7$
$> ./pgcd 17 3 | cat -e
1$
$> ./pgcd | cat -e
$
==========================================================
================================
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char const *argv[])
{
    int nbr1;
    int nbr2;

    if (argc == 3)
    {
        if ((nbr1 = atoi(argv[1])) > 0 && (nbr2 =
atoi(argv[2])) > 0)
        {
            while (nbr1 != nbr2)
            {
                if (nbr1 > nbr2)
                    nbr1 -= nbr2;
                else
                    nbr2 -= nbr1;
            }
            printf("%d", nbr1);
        }
    }
    printf("\n");
    return (0);
}
==
#include <stdio.h>
#include <stdlib.h>

void pgcd(int nb1, int nb2)
{
    int div;
    int pgcd;

    div = 1;
    if (nb1 <= 0 || nb2 <= 0)
        return ;
    while (div <= nb1 || div <= nb2)
    {
        if (nb1 % div == 0 & nb2 % div == 0)
            pgcd = div;
        div++;
    }
    printf("%d", pgcd);
}

int main(int argc, char **argv)
{
    if (argc == 3)
        pgcd(atoi(argv[1]), atoi(argv[2]));
    printf("\n");
```

```
	return (0);
}
```

```
======================================./3-0-
print_hex.txt=========================================
Assignment name  : print_hex
Expected files   : print_hex.c
Allowed functions: write
-----------------------------------------------------------
----------------------

Write a program that takes a positive (or zero) number
expressed in base 10,
and displays it in base 16 (lowercase letters) followed by
a newline.

If the number of parameters is not 1, the program displays
a newline.

Examples:

$> ./print_hex "10" | cat -e
a$
$> ./print_hex "255" | cat -e
ff$
$> ./print_hex "5156454" | cat -e
4eae66$
$> ./print_hex | cat -e
$
===========================================================
================================
```

```c
#include <unistd.h>

int ft_atoi(char *s)
{
	long r;
	int sign;

	while(*s == 32 || (*s >= 9 && *s <= 13))
		s++;
	sign = (*s == '-') ? -1 : 1;
	(*s == '-' || *s == '+') ? s++ : s;
	r = 0;
```

```
    while (*s >= '0' && *s <= '9')
        r = r * 10 + *s++ - '0';
    return ((int)r * sign);
}

void print_hex(int n)
{
    if (n >= 16)
        print_hex(n / 16);
    n = n % 16;
    n += n < 10 ? '0' : 'a' - 10;
    write(1, &n, 1);
}

int main(int ac, char **av)
{
    if (ac == 2)
        print_hex(ft_atoi(av[1]));
    write(1, "\n", 1);
    return (1);
}
```

==

```
#include <unistd.h>

void print_hex(int p)
{
    char *str;

    str = "0123456789abcdef";
    if (p == 0)
        write (1, "0", 1);
    while (p)
    {
        write(1, &str[p % 16], 1);
        p  /= 16;
    }
}

int ft_atoi(char *str)
{
    int i;
    int nbr;
    int sign;

    i = 0;
    nbr = 0;
```

```c
    sign = 1;
    if (!str[i])
        return (0);
    while (str[i] == ' ' || (*str >= 9 && *str <= 13))
        i += 1;
    if (str[i] == '-' || str[i] == '+')
        if (str[i++] == '-')
            sign *= -1;
    while (str[i] && (str[i] >= '0' && '9' >= str[i]))
        nbr = (nbr * 10) + str[i++] - '0';
    return (nbr * sign);
}

int main(int argc, char *argv[])
{
    if (argc == 2)
        print_hex(ft_atoi(argv[1]));
    write(1, "\n", 1);
    return (0);
}
```

```
======================================./3-0-
rstr_capitalizer.txt======================================
===
Assignment name  : rstr_capitalizer
Expected files   : rstr_capitalizer.c
Allowed functions: write
----------------------------------------------------------
----------------------
```

Write a program that takes one or more strings and, for each argument, puts the last character of each word (if it's a letter) in uppercase and the rest in lowercase, then displays the result followed by a \n.

A word is a section of string delimited by spaces/tabs or the start/end of the string. If a word has a single letter, it must be capitalized.

If there are no parameters, display \n.

```
Examples:

$> ./rstr_capitalizer | cat -e
$
$> ./rstr_capitalizer "Premier PETIT TesT" | cat -e
premieR petiT tesT$
$> ./rstr_capitalizer "DeuxiEmE tEST uN PEU moinS  facile"
"   attention C'EST pas dur QUAND mEmE" "ALLer UN DeRNier
0123456789pour LA rouTE    E " | cat -e
deuxiemE tesT uN peU moinS  facilE$
   attentioN c'esT paS duR quanD memE$
alleR uN dernieR 0123456789pouR lA routE    E $
$>
==========================================================
================================
```

```
#include <unistd.h>

void    rstr_capitalizer(char *str)
{
    int i;

    i = 0;
    while (str[i])
    {
        if (str[i] >= 'A' && str[i] <= 'Z')
            str[i] += 32;
        if ((str[i] >= 'a' && str[i] <= 'z') && (str[i + 1] == '
' \
                    || str[i + 1] == '\t' || str[i + 1] ==
'\0'))
            str[i] -= 32;
        write(1, &str[i++], 1);
    }
}

int main(int ac, char **av)
{
    int i;

    if (ac < 1)
        write(1, "\n", 1);
    else
    {
        i = 1;
        while (i < ac)
        {
            rstr_capitalizer(av[i]);
            write(1, "\n", 1);
            i += 1;
        }
    }
    return (0);
}
```

```
======================================./3-1-
expand_str.txt=========================================
Assignment name  : expand_str
Expected files   : expand_str.c
Allowed functions: write
----------------------------------------------------------
----------------------
```

Write a program that takes a string and displays it with exactly three spaces
between each word, with no spaces or tabs either at the beginning or the end,
followed by a newline.

A word is a section of string delimited either by spaces/tabs, or by the
start/end of the string.

If the number of parameters is not 1, or if there are no words, simply display
a newline.

Examples:

```
$> ./expand_str "vous   voyez   c'est   facile
d'afficher   la   meme   chose" | cat -e
vous   voyez   c'est   facile   d'afficher   la   meme
chose$
$> ./expand_str " seulement          la c'est      plus
dur " | cat -e
seulement   la   c'est   plus   dur$
$> ./expand_str "comme c'est cocasse" "vous avez entendu,
Mathilde ?" | cat -e
```

```
$
$> ./expand_str "" | cat -e
$
$>
==========================================================
================================
```

```
#include <unistd.h>

int main(int argc, char const *argv[])
{
    int i;
    int flag;

    if (argc == 2)
    {
        i = 0;
        while (argv[1][i] == ' ' || argv[1][i] == '\t')
            i++;
        while (argv[1][i])
        {
            if (argv[1][i] == ' ' || argv[1][i] == '\t')
                flag = 1;
            if (!(argv[1][i] == ' ' || argv[1][i] == '\t'))
            {
                if (flag)
                    write(1, "   ", 3);
                flag = 0;
                write(1, &argv[1][i], 1);
            }
            i++;
        }
    }
    write(1, "\n", 1);
    return (0);
}
```

```
======================================./3-1-
tab_mult.txt=========================================
Assignment name  : tab_mult
Expected files   : tab_mult.c
Allowed functions: write
----------------------------------------------------------
----------------------
```

Write a program that displays a number's multiplication table.

The parameter will always be a strictly positive number that fits in an int,
and said number times 9 will also fit in an int.

If there are no parameters, the program displays \n.

Examples:

```
$>./tab_mult 9
1 x 9 = 9
2 x 9 = 18
3 x 9 = 27
4 x 9 = 36
5 x 9 = 45
6 x 9 = 54
7 x 9 = 63
8 x 9 = 72
9 x 9 = 81
$>./tab_mult 19
1 x 19 = 19
2 x 19 = 38
3 x 19 = 57
4 x 19 = 76
5 x 19 = 95
6 x 19 = 114
7 x 19 = 133
8 x 19 = 152
9 x 19 = 171
$>
$>./tab_mult | cat -e
$
$>
==========================================================
================================
```

```
#include <unistd.h>

int  ft_atoi(char *str)
{
    int result;
    int sign;
```

```
     result = 0;
     sign = 1;
     while (*str == ' ' || (*str >= 9 && *str <= 13))
          str++;
     if (*str == '-')
          sign = -1;
     if (*str == '-' || *str == '+')
          str++;
     while (*str >= '0' && *str <= '9')
     {
          result = result * 10 + *str - '0';
          str++;
     }
     return (sign * result);
}

void ft_putchar(char c)
{
     write(1, &c, 1);
}

void ft_putnbr(int nb)
{
     if (nb == -2147483648)
     {
          ft_putchar('-');
          ft_putchar('2');
          nb = (nb % 1000000000 * -1);
     }
     if (nb < 0)
     {
          ft_putchar('-');
          nb = (nb * -1);
     }
     if (nb / 10 > 0)
          ft_putnbr(nb / 10);
     ft_putchar(nb % 10 + '0');
}

int main(int argc, char *argv[])
{
     int  i;
     int  nbr;

     if (argc != 2)
          write(1, "\n", 1);
```

```
    else
    {
        i = 1;
        nbr = ft_atoi(argv[1]);
        while (i <= 9)
        {
            ft_putnbr(i);
            write(1, " x ", 3);
            ft_putnbr(nbr);
            write(1, " = ", 3);
            ft_putnbr(i * nbr);
            write(1, "\n", 1);
            i += 1;
        }
    }
    return (0);
}
```

```
=========================================./3-2-
ft_atoi_base.txt=========================================
Assignment name  : ft_atoi_base
Expected files   : ft_atoi_base.c
Allowed functions: None
----------------------------------------------------------------
---------------------

Write a function that converts the string argument str
(base N <= 16)
to an integer (base 10) and returns it.

The characters recognized in the input are:
0123456789abcdef
Those are, of course, to be trimmed according to the
requested base. For
example, base 4 recognizes "0123" and base 16 recognizes
"0123456789abcdef".

Uppercase letters must also be recognized: "12fdb3" is the
same as "12FDB3".

Minus signs ('-') are interpreted only if they are the
first character of the
string.
```

Your function must be declared as follows:

```
int  ft_atoi_base(const char *str, int str_base);
===========================================================
================================
```

```c
int ft_isblank(char c)
{
     if (c <= 32)
          return (1);
     return (0);
}

int       ft_isvalid(char c, int base)
{
     char digits[17] = "0123456789abcdef";
     char digits2[17] = "0123456789ABCDEF";

     while (base--)
          if (digits[base] == c || digits2[base] == c)
               return (1);
     return (0);
}

int       ft_value_of(char c)
{
     if (c >= '0' && c <= '9')
          return (c - '0');
     else if (c >= 'a' && c <= 'f')
          return (c - 'a' + 10);
     else if (c >= 'A' && c <= 'F')
          return (c - 'A' + 10);
     return (0);
}

int       ft_atoi_base(const char *str, int str_base)
{
     int result;
     int sign;

     result = 0;
     while (ft_isblank(*str))
          str++;
     sign = (*str == '-') ? -1 : 1;
     (*str == '-' || *str == '+') ? ++str : 0;
     while (ft_isvalid(*str, str_base))
```

```
        result = result * str_base + ft_value_of(*str++);
    return (result * sign);
}
===
#include <stdio.h>
#include <stdlib.h>

int ft_atoi_base(const char *str, int base);

int     main(void)
{
    printf("%d\n", ft_atoi_base("011", atoi("2")));
    printf("%d\n", ft_atoi_base("16", atoi("8")));
    printf("%d\n", ft_atoi_base("123", atoi("10")));
    printf("%d\n", ft_atoi_base("FF", atoi("16")));
}
```

```
======================================./3-3-
ft_range.txt=========================================
Assignment name  : ft_range
Expected files   : ft_range.c
Allowed functions: malloc
----------------------------------------------------------
----------------------

Write the following function:

int     *ft_range(int start, int end);

It must allocate (with malloc()) an array of integers,
fill it with consecutive
values that begin at start and end at end (Including start
and end !), then
return a pointer to the first value of the array.

Examples:

- With (1, 3) you will return an array containing 1, 2 and
3.
- With (-1, 2) you will return an array containing -1, 0,
1 and 2.
- With (0, 0) you will return an array containing 0.
- With (0, -3) you will return an array containing 0, -1,
-2 and -3.
=================================================================
====================================
```

```c
#include <stdlib.h>

int  *ft_range(int min, int max)
{
     int  n;
     int  *s;

     n = max >= min ? max - min : min - max;
     if (!(s= (int *)malloc(sizeof(int) * (n))))
          return (NULL);
     while (max != min)
          *s++ = max > min ? min++ : min--;
     *s = min;
     return (s - n);
}
==
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int  *ft_range(int min, int max);

void ft_putchar(char c)
{
     write(1, &c, 1);
     return;
}

void ft_putnbr(int n)
{
     if (n > 2147483647 || n < -2147483648)
          return;
     if (n == -2147483648)
     {
          write(1, "-2147483648", 12);
          return;
     }
     if (n < 0)
     {
          n *= -1;
          ft_putchar('-');
     }
     if (n < 10)
     {
          ft_putchar(n + '0');
          return;
     }
     ft_putnbr(n / 10);
     ft_putchar((n % 10) + '0');
     return;
}

int  ft_atoi(char *str)
{
     int result;
     int sign;

     result = 0;
     sign = 1;
     while (*str == ' ' || (*str >= 9 && *str <= 13))
          str++;
     if (*str == '-')
          sign = -1;
     if (*str == '-' || *str == '+')
          str++;
     while (*str >= '0' && *str <= '9')
     {
          result = result * 10 + *str - '0';
          str++;
     }
     return (sign * result);
}

int  main(int ac, char **av)
{
     int  *s;
     int  n;
     int  min;
     int  max;

     if (ac != 3)
          return (0);
     min = ft_atoi(av[1]);
     max = ft_atoi(av[2]);
     n = max >= min ? max - min + 1 : min - max + 1;

     s = ft_range(min, max);
     while (*s && n--)
     {
          ft_putnbr(*s++);
          ft_putchar('\n');
     }
     return (1);
}
```

```
======================================./3-4-
paramsum.txt=========================================
Assignment name  : paramsum
Expected files   : paramsum.c
Allowed functions: write
----------------------------------------------------------
----------------------

Write a program that displays the number of arguments
passed to it, followed by
a newline.

If there are no arguments, just display a 0 followed by a
newline.

Example:

$>./paramsum 1 2 3 5 7 24
6
$>./paramsum 6 12 24 | cat -e
3$
$>./paramsum | cat -e
0$
$>
==========================================================
================================
```

```
#include <unistd.h>

void ft_putchar(char c)
{
     write(1, &c, 1);
}

void ft_putnbr(int n)
{
     (n < 0 ? ft_putchar('-') : 1);
```

```c
    n *= (n > 0 ? -1 : 1);
    (n <= -10 ? ft_putnbr(-(n / 10)) : 1);
    ft_putchar('0' - n % 10);
}

int  main(int ac, char **av)
{
    (void)av;
    ft_putnbr(ac - 1);
    write(1, "\n", 1);
    return (0);
}
```

```
======================================./3-4-
str_capitalizer.txt=======================================
==
Assignment name  : str_capitalizer
Expected files   : str_capitalizer.c
Allowed functions: write
----------------------------------------------------------
----------------------

Write a program that takes one or several strings and, for
each argument,
capitalizes the first character of each word (If it's a
letter, obviously),
puts the rest in lowercase, and displays the result on the
standard output,
followed by a \n.

A "word" is defined as a part of a string delimited either
by spaces/tabs, or
by the start/end of the string. If a word only has one
letter, it must be
capitalized.

If there are no arguments, the progam must display \n.

Example:

$> ./str_capitalizer | cat -e
$
$> ./str_capitalizer "Premier PETIT TesT" | cat -e
```

```
Premier Petit Test$
$> ./str_capitalizer "DeuxiEmE tEST uN PEU moinS  facile"
"   attention C'EST pas dur QUAND mEmE" "ALLer UN DeRNier
0123456789pour LA rouTE    E " | cat -e
Deuxieme Test Un Peu Moins  Facile$
   Attention C'est Pas Dur Quand Meme$
Aller Un Dernier 0123456789pour La Route    E $
$>
==========================================================
================================
```

```c
#include <unistd.h>

void str_capitalizer(char *str)
{
     int       i;

     i = 0;
     if (str[i] >= 'a' && 'z' >= str[i])
          str[i] -= 32;
     write(1, &str[i], 1);
     while (str[++i])
     {
          if (str[i] >= 'A' && 'Z' >= str[i])
               str[i] += 32;
          if ((str[i] >= 'a' && 'z' >= str[i]) && (str[i - 1] ==
' ' || \
          str[i - 1] == '\t'))
               str[i] -= 32;
          write(1, &str[i], 1);
     }
}

int       main(int argc, char *argv[])
{
     int       i;

     if (argc < 2)
          write(1, "\n", 1);
     else
     {
          i = 1;
          while (i < argc)
          {
               str_capitalizer(argv[i]);
               write(1, "\n", 1);
               i += 1;
```

```
            }
        }
        return (0);
}
```

==./3-4-lcm.txt==

Assignment name : lcm
Expected files   : lcm.c
Allowed functions:

Write a function who takes two unsigned int as parameters and returns the computed LCM of those parameters.

LCM (Lowest Common Multiple) of two non-zero integers is the smallest postive integer divisible by the both integers.

A LCM can be calculated in two ways:

- You can calculate every multiples of each integers until you have a common multiple other than 0

- You can use the HCF (Highest Common Factor) of these two integers and calculate as follows:

```
        LCM(x, y) = | x * y | / HCF(x, y)
```

| x * y | means "Absolute value of the product of x by y"

If at least one integer is null, LCM is equal to 0.

Your function must be prototyped as follows:

```
  unsigned int    lcm(unsigned int a, unsigned int b);
```

==

```
================================
unsigned int   lcm(unsigned int a, unsigned int b)
{
     int  gcd;
     int  org_a;
     int  org_b;

     gcd = 0;
     org_a = a;
     org_b = b;
     while (1)
     {
          if (a == 0)
               break;
          b %=a;
          if (b == 0)
               break;
          a %= b;
     }
     gcd = (!b) ? a : b;
     return (gcd ? (org_a / gcd * org_b) : 0);
}
==
#include <stdio.h>

unsigned int   lcm(unsigned int a, unsigned int b);

int  main(void)
{
     printf("%d\n", lcm(122, 22));
     printf("%d\n", lcm(100, 10));
     printf("%d\n", lcm(4242, 42));
     printf("%d\n", lcm(5, 9));
     return (0);
}
```

```
======================================./4-0-
fprime.txt=========================================
Assignment name  : fprime
Expected files   : fprime.c
Allowed functions: printf, atoi
-----------------------------------------------------------------
------------------------
```

Write a program that takes a positive int and displays its prime factors on the standard output, followed by a newline.

Factors must be displayed in ascending order and separated by '*', so that the expression in the output gives the right result.

If the number of parameters is not 1, simply display a newline.

The input, when there's one, will be valid.

Examples:

```
$> ./fprime 225225 | cat -e
3*3*5*5*7*11*13$
$> ./fprime 8333325 | cat -e
3*3*5*5*7*11*13*37$
$> ./fprime 9539 | cat -e
9539$
$> ./fprime 804577 | cat -e
804577$
$> ./fprime 42 | cat -e
2*3*7$
$> ./fprime 1 | cat -e
1$
$> ./fprime | cat -e
$
$> ./fprime 42 21 | cat -e
$
```

==
================================

```c
#include <stdlib.h>
#include <stdio.h>

void fprime(unsigned int nb)
{
    unsigned  i;

    if (nb == 1)
        printf("1");
    else
```

```
     {
          i = 1;
          while (nb > 1)
          {
               if (nb % ++ i == 0)
               {
                    printf("%d", i);
                    nb /= i;
                    if (nb > 1)
                         printf("*");
                    --i;
               }
          }
     }
}

int  main(int ac, char **av)
{
     if (ac == 2 && *av[1])
          fprime(atoi(av[1]));
     printf("\n");
     return (0);
}

===
#include <stdlib.h>
#include <stdio.h>

/* Recursive way to do fprime */

void fprime(int nb, int a, int i)
{
     a++;
     while (nb % i != 0 && i < nb)
          i++;
     if (nb % i == 0)
     {
          if (a != 1)
               printf("*");
          printf("%d", i);
          if (nb != i)
               fprime(nb / i, a, i);
     }
     else
          printf("%d", nb);
}
```

```
int  main(int ac, char **av)
{
     if (ac == 2 && av[1][0] != '\0')
          fprime(atoi(av[1]), 0, 2);
     printf("\n");
     return (0);
}
===
#include <stdio.h>
#include <stdlib.h>

int  main(int argc, char *argv[])
{
     int  i;
     int  nbr;

     if (argc == 2)
     {
          i = 1;
          nbr = atoi(argv[1]);
          if (nbr == 1)
               printf("1");
          while (nbr >= ++i)
          {
               if (nbr % i == 0)
               {
                    printf("%d", i);
                    if (nbr == i)
                         break ;
                    printf("*");
                    nbr /= i;
                    i = 1;
               }
          }
     }
     printf("\n");
     return (0);
}
```

```
======================================./4-0-
ft_list_foreach.txt=======================================
==
Assignment name  : ft_list_foreach
Expected files   : ft_list_foreach.c, ft_list.h
```

Allowed functions:

Write a function that takes a list and a function pointer, and applies this function to each element of the list.

It must be declared as follows:

```
void    ft_list_foreach(t_list *begin_list, void (*f)(void
*));
```

The function pointed to by f will be used as follows:

```
(*f)(list_ptr->data);
```

You must use the following structure, and turn it in as a file called ft_list.h:

```
typedef struct    s_list
{
    struct s_list *next;
    void          *data;
}                 t_list;
```

===
================================

```
#include "4-0-ft_list.h"

void ft_list_foreach(t_list *begin_list, void (*f)(void *))
{
     t_list *list_ptr;

     list_ptr = begin_list;
     while (list_ptr)
     {
          (*f)(list_ptr->data);
          list_ptr = list_ptr->next;
     }
}
```

==

```
#include <stdio.h>
```

```c
#include <stdlib.h>
#include "4-0-ft_list.h"

void ft_list_foreach(t_list *begin_list, void (*f)(void *));

void print_data(void *data)
{
     printf("%s\n", data);
}

int  main(void)
{
     t_list *test_list = malloc(sizeof(t_list));
     test_list -> data = "what up";
     test_list -> next = malloc(sizeof(t_list));
     test_list -> next -> data = "42";
     test_list -> next -> next = malloc(sizeof(t_list));
     test_list -> next -> next -> data = "peeps?";
     test_list -> next -> next -> next = NULL;

     ft_list_foreach(test_list, print_data);
     return (0);
}
```

```
======================================./4-0-
ft_split.txt=========================================
Assignment name  : ft_split
Expected files   : ft_split.c
Allowed functions: malloc
----------------------------------------------------------
----------------------

Write a function that takes a string, splits it into
words, and returns them as
a NULL-terminated array of strings.

A "word" is defined as a part of a string delimited either
by spaces/tabs/new
lines, or by the start/end of the string.

Your function must be declared as follows:

char    **ft_split(char *str);
```

```
=================================================================
====================================

#include <stdlib.h>

int  ft_isspace(char c)
{
     return ((c == ' ' || (c >= 9 && c <= 13)) ? 1 : 0);
}

static int     r_size(char *s)
{
     unsigned int len;

     len = 0;
     while (*s)
     {
          if (ft_isspace(*s))
               ++s;
          else
          {
               ++len;
               while (*s && !ft_isspace(*s))
                    ++s;
          }
     }
     return (len);
}

char           **ft_split(char *s)
{
     int       i = 0;
     int       j = 0;
     int       k;
     char **r;
     int       w_len = 0;

     if (!(r = (char **)malloc(sizeof(char*) * (r_size(s) +
1))))
          return (0);
     while (s[i] && j < r_size(s))
     {
          while (s[i] && ft_isspace(s[i]))
               i++;
          while (s[i] && !ft_isspace(s[i]))
          {
               w_len++;
               i++;
          }
          if (!(r[j] = (char *)malloc(sizeof(char) * (w_len +
1))))
               return (0);
          k = 0;
          while (w_len)
               r[j][k++] = s[i - w_len--];
          r[j++][k] = '\0';
     }
     return (r);
}

===
#include <stdio.h>

char           **ft_split(char *s);

int  main(void)
{
     int  i = 0;
     char **split_me;

     split_me = ft_split("I dare you to split me!");
     while (i < 6)
     {
          printf("Word %d: %s\n", i, split_me[i]);
          i++;
     }
     return (0);
}
```

=====================================./4-1-rev_wstr.txt===

Assignment name : rev_wstr
Expected files   : rev_wstr.c
Allowed functions: write, malloc, free

Write a program that takes a string as a parameter, and prints its words in reverse order.

A "word" is a part of the string bounded by spaces and/or tabs, or the begin/end of the string.

If the number of parameters is different from 1, the program will display '\n'.

In the parameters that are going to be tested, there won't be any "additional" spaces (meaning that there won't be additionnal spaces at the beginning or at the end of the string, and words will always be separated by exactly one space).

Examples:

```
$> ./rev_wstr "le temps du mepris precede celui de l'indifference" | cat -e
l'indifference de celui precede mepris du temps le$
$> ./rev_wstr "abcdefghijklm"
abcdefghijklm
$> ./rev_wstr "il contempla le mont" | cat -e
mont le contempla il$
$> ./rev_wstr | cat -e
$
$>
==========================================================
================================
```

```
#include <unistd.h>

int  ft_isblank(char c)
{
     if (c == ' ' || c == '\t')
          return (1);
     return (0);
}

void rev_wstr(char *s)
{
     int  wc = 0;
     int  i = 0;
     int  len;
     int  a;
     while (s[i])
          if (!ft_isblank(s[i++]) && (!wc || ft_isblank(s[i -
2])))
               ++wc;
     while (s[--i])
     {
          if (!ft_isblank(s[i]) && wc--)
          {
               a = 0;
               len = 1;
               while (s[i - 1] && !ft_isblank(s[i - 1]) && +
+len)
                    --i;
               while (len-- && write(1, &s[i + a++], 1));
               (wc) ? write(1, " ", 1) : 0;
          }
     }
}

int  main(int ac, char **av)
{
     if (ac == 2 && *av[1])
          rev_wstr(av[1]);
     write(1, "\n", 1);
     return (0);
}
```

```
======================================./4-2-
ft_list_remove_if.txt=====================================
====
Assignment name  : ft_list_remove_if
Expected files   : ft_list_remove_if.c
Allowed functions: free
----------------------------------------------------------
----------------------
```

Write a function called ft_list_remove_if that removes from the passed list any element the data of which is "equal" to the reference data.

It will be declared as follows :

```
void ft_list_remove_if(t_list **begin_list, void
*data_ref, int (*cmp)());
```

cmp takes two void* and returns 0 when both parameters are equal.

You have to use the ft_list.h file, which will contain:

```
$>cat ft_list.h
typedef struct      s_list
{
    struct s_list   *next;
    void            *data;
}                   t_list;
$>
==========================================================
================================
```

```
#include <stdlib.h>
#include "4-2-ft_list.h"

void ft_list_remove_if(t_list **begin_list, void *data_ref, int
(*cmp)())
{
    t_list    *to_free;

    if (*begin_list)
    {
        if (cmp((*begin_list)->data, data_ref) == 0)
        {
            to_free = *begin_list;
            *begin_list = (*begin_list)->next;
            free(to_free);
            ft_list_remove_if(begin_list, data_ref, cmp);
        }
        else
            ft_list_remove_if(&(*begin_list)->next, data_ref,
cmp);
    }
}
==
#include <stdlib.h>
#include <unistd.h>
#include "4-2-ft_list.h"
```

```
void ft_list_remove_if(t_list **begin_list, void *data_ref, int
(*cmp)());

void ft_putstr(char *str)
{
	int  i;

	i = 0;
	while (*str)
	{
		write(1, &*str, 1);
		str++;
	}
}

void print_list(t_list *list)
{
	while (list)
	{
		ft_putstr(list->data);
		ft_putstr("\n");
		list = list->next;
	}
}

int  cmp(char *elem1, char *elem2)
{
	int  i;

	i = 0;
	while (elem1[i] != '\0' && elem2[i] != '\0' && elem1[i] ==
elem2[i])
		i++;
	if (elem1[i] == elem2[i])
		return (0);
	return (1);
}

int  main(void)
{
	t_list *whine_list = malloc(sizeof(t_list));

	whine_list -> data = "C sucks";
	whine_list -> next = malloc(sizeof(t_list));
	whine_list -> next -> data = "Python is pitiful";
	whine_list -> next -> next = malloc(sizeof(t_list));
```

```
    whine_list -> next -> next -> data = "Ruby's raunchy";
    whine_list -> next -> next -> next =
malloc(sizeof(t_list));
    whine_list -> next -> next -> next -> data = "Wish I was
using lisp lists";
    whine_list -> next -> next -> next -> next = NULL;

    ft_list_remove_if(&whine_list, "C sucks", &cmp);
    print_list(whine_list);
    return (0);
}
```

```
======================================./4-3-
sort_int_tab.txt=========================================
Assignment name  : sort_int_tab
Expected files   : sort_int_tab.c
Allowed functions:
----------------------------------------------------------
----------------------
```

Write the following function:

```
void sort_int_tab(int *tab, unsigned int size);
```

It must sort (in-place) the 'tab' int array, that contains exactly 'size' members, in ascending order.

Doubles must be preserved.

Input is always coherent.

```
==========================================================
================================
void sort_int_tab(int *tab, unsigned int size)
{
    unsigned int   i;
    int  temp;

    i = 0;
    while (i < (size - 1))
    {
        if (tab[i] > tab[i + 1])
        {
               temp = tab[i];
               tab[i] = tab[i+ 1];
               tab[i + 1] = temp;
               i = 0;
          }
          else
               i++;
     }
}
==
#include <stdio.h>

void sort_int_tab(int *tab, unsigned int size);

int       main(void)
{
     int a[6] = {9, 7, 6, 4, 5, 10};
     int i = 0;
     int size = 6;

     sort_int_tab(a, size);
     while (i < size)
          printf("%d\n", a[i++]);
     return (0);
}
```

```
==========================================./4-3-sort_list.txt=============================================
Assignment name  : sort_list
Expected files   : sort_list.c
Allowed functions:
----------------------------------------------------------------------------------------
```

Write the following functions:

```
t_list   *sort_list(t_list* lst, int (*cmp)(int, int));
```

This function must sort the list given as a parameter, using the function pointer cmp to select the order to apply, and returns a pointer to the first element of the sorted list.

Duplications must remain.

Inputs will always be consistent.

You must use the type t_list described in the file list.h
that is provided to you. You must include that file
(#include "list.h"), but you must not turn it in. We will
use our own
to compile your assignment.

Functions passed as cmp will always return a value
different from
0 if a and b are in the right order, 0 otherwise.

For example, the following function used as cmp will sort
the list
in ascending order:

```
int ascending(int a, int b)
{
    return (a <= b);
}
==========================================================
================================
```

```
#include <stdlib.h>
#include "list.h"

t_list    *sort_list(t_list *lst, int (*cmp)(int, int))
{
    int  swap;
    t_list    *tmp;

    tmp = lst;
    while(lst->next != NULL)
    {
        if (((*cmp)(lst->data, lst->next->data)) == 0)
        {
            swap = lst->data;
            lst->data = lst->next->data;
            lst->next->data = swap;
            lst = tmp;
        }
        else
            lst = lst->next;
```

```
     }
     lst = tmp;
     return (lst);
}

==
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

t_list    *sort_list(t_list *lst, int (*cmp)(int, int));

t_list    *add_int(t_list *list, int nb)
{
     t_list *new;

     new = (t_list*)malloc(sizeof(t_list));
     new->data = nb;
     new->next = list;
     return (new);
}

int       ascending(int a, int b)
{
          return (a <= b);
}

int  main(void)
{
     t_list *list;

     list = NULL;
     list = add_int(list, 9);
     list = add_int(list, 3);
     list = add_int(list, 2);
     list = add_int(list, 4);
     list = add_int(list, 1);
     list = sort_list(list, &ascending);

     while(list != NULL)
     {
          printf("%d\n", list->data);
          list = list->next;
     }

     return (0);
}
```

```
======================================./4-4-
ft_itoa.txt=========================================
Assignment name  : ft_itoa
Expected files   : ft_itoa.c
Allowed functions: malloc
----------------------------------------------------------
----------------------

Write a function that takes an int and converts it to a
null-terminated string.
The function returns the result in a char array that you
must allocate.

Your function must be declared as follows:

char *ft_itoa(int nbr);
==========================================================
================================
```

```c
#include <stdlib.h>

char	*ft_itoa(int nbr);

char *ft_itoa(int nbr)
{
	char *str;
	char *t;
	char *u;

	if (!(str = (char *)malloc(16)))
		return (NULL);
	t = str;
	(nbr < 0 ? *t++ = '-' : 1);
	if (nbr > 0)
		nbr = -nbr;
	if (nbr <= -10)
	{
		u = ft_itoa(-(nbr / 10));
		while (*u)
			*t++ = *u++;
	}
```

```
     *t = '0' - nbr % 10;
     *(t + 1) = '\0';
     return (str);
}
===
#include <stdlib.h>

char    *ft_itoa(int nbr)
{
     int       len;
     long n_tmp;
     char *str;

     len = 0;
     n_tmp = nbr;
     if (nbr == -2147483648)
          return ("-2147483648");
     if (!(str = (char *)malloc(sizeof(char) * len + 1)))
          return (NULL);
     str[len] ='\0';
     if (nbr == 0)
     {
          str[0] = '0';
          return (str);
     }
     if (nbr < 0)
     {
          len += 1;
          nbr *= -1;
          str[0] = '-';
     }
     while (n_tmp)
     {
          n_tmp /= 10;
          len += 1;
     }
     while (nbr)
     {
          str[--len] = (nbr % 10) + '0';
          nbr /= 10;
     }
     return (str);
}
===
#include <stdio.h>

char        *ft_itoa(int n);
```

```c
int main(void)
{
     printf("%s\n", ft_itoa(33));
     printf("%s\n", ft_itoa(-33));
     printf("%s\n", ft_itoa(12345));
     printf("%s\n", ft_itoa(-12345));
     printf("%s\n", ft_itoa(98765));
     printf("%s\n", ft_itoa(-98765));
     printf("%s\n", ft_itoa(45));
     printf("%s\n", ft_itoa(-45));
     printf("%s\n", ft_itoa(-2147483648));
     printf("%s\n", ft_itoa(2147483647));
     printf("%s\n", ft_itoa(0));
     return (0);
}
```

```
====================================./4-5-check_mate.txt==========================================
Assignment name  : checkmate
Expected files   : *.c, *.h
Allowed functions: write, malloc, free
--------------------------------------------------------------------------------
```

Write a program who takes rows of a chessboard in argument and check if your King is in a check position.

Chess is played on a chessboard, a squared board of 8-squares length with specific pieces on it : King, Queen, Bishop, Knight, Rook and Pawns.
For this exercice, you will only play with Pawns, Bishops, Rooks and Queen...
and obviously a King.

Each piece have a specific method of movement, and all patterns of capture are detailled in the examples.txt file.

A piece can capture only the first ennemy piece it founds on its capture patterns.

The board have a variable size but will remains a square. There's only one King and all other pieces are against it. All other characters except those used for pieces are considered as empty squares.

The King is considered as in a check position when an other enemy piece can capture it. When it's the case, you will print "Success" on the standard output followed by a newline, otherwise you will print "Fail" followed by a newline.

If there is no arguments, the program will only print a newline.

Examples:

```
$> ./check_mate '..' '.K' | cat -e
Fail$
$> ./check_mate 'R...' '.K..' '..P.' '....' | cat -e
Success$
$> ./check_mate 'R...' 'iheK' '....' 'jeiR' | cat -e
Success$
$> ./check_mate | cat -e
$
$>
```

****

Some subject.en.txts on the web have this example:

```
$> ./chessmate 'R...' '..P.' '.K..' '....' | cat -e
Success$
```

Which would indicate that checks need to be down both ways.

Most solutions will:

```
Fail$
```

As they are only checking in one direction.

****

==
================================

```
#include "4-5-check_mate-02.h"

size_t    ft_strlen(char *s)
{
    size_t    i;

    i = 0;
    while (s[i])
        i++;
    return (i);
}


int  ft_opiece(char piece)
{
    if (piece == 'P' || piece == 'Q' || piece == 'B' || piece
== 'R')
        return (1);
    return (0);
}

/**** Pawn ****/
int  ft_pawn(char **board, int y, int x)
{
    if (y > 1)
    {
        if (board[y - 1][x - 1] == 'K')
            return (1);
        else if (board[y -1][x + 1] == 'K')
            return (1);
    }
    return (0);
}
/*end-pawn*/

/**** Rook ****/
int  ft_rook(char **board, int y, int x)
{
    int  len;
    int  j;

    len = (int)ft_strlen(board[y]);
    j = x;
    while (++j < len && ft_opiece(board[y][j]) != 1) //
Horizontal++
    {
        if (board[y][j] == 'K')
            return (1);
    }
    j = x;
    while (--j >= 0 && ft_opiece(board[y][j]) != 1) //
```

```
Horizontal--
     {
          if (board[y][j] == 'K')
               return (1);
     }
     j = y;
     while (++j <= len && ft_opiece(board[j][x]) != 1) //
Vertical--
     {
          if (board[j][x] == 'K')
               return (1);
     }
     j = y;
     while (--j >= 1 && ft_opiece(board[j][x]) != 1) //Vertical+
+
     {
          if (board[j][x] == 'K')
               return (1);
     }
     return (0);
}
/*end-rook*/

/**** Bishop ****/
int  ft_bishop(char **board, int y, int x)
{
     int  len;
     int  i;
     int  j;

     len = (int)ft_strlen(board[1]);
     i = y;
     j = x;
     while (++i <= len && ++j < len && ft_opiece(board[i][j]) !=
1) //Down Right
     {
          if (board[i][j] == 'K')
               return (1);
     }
     i = y;
     j = x;
     while (--i >= 1 && --j >= 0 && ft_opiece(board[i][j]) !=
1) //Down Left
     {
          if (board[i][j] == 'K')
               return (1);
     }
     i = y;
     j = x;
     while (--i >= 1 && ++j < len && ft_opiece(board[i][j]) !=
1) //Up Right
     {
          if (board[i][j] == 'K')
               return (1);
     }
     i = y;
     j = x;
     while (--i >= 1 && --j >= 0 && ft_opiece(board[i][j]) !=
1) //Up Left
     {
          if (board[i][j] == 'K')
               return (1);
     }
     return (0);
}
/*end-bishop*/

static int     ft_checkmate(char **av)
{
     int  i;
     int j;

     i = 1;
     while (av[i])
     {
          j = 0;
          while (av[i][j])
          {
               if (av[i][j] == 'R' && ft_rook(av, i, j) == 1)
                    return (1);
               if (av[i][j] == 'P' && ft_pawn(av, i, j) == 1)
                    return (1);
               if (av[i][j] == 'B' && ft_bishop(av, i, j) == 1)
                    return (1);
               if (av[i][j] == 'Q' && (ft_bishop(av, i, j) == 1
|| ft_rook(av, i, j) == 1))
                    return (1);
               j++;
          }
          i++;
     }
     return (0);
}
```

```c
int main (int ac, char **av)
{
     if (ac > 1 && ac == (int)(ft_strlen(av[1]) + 1))
     {
          int  i;

          i = 1;
          while (av[i] != NULL)
          {
               if (((int)ft_strlen(av[i]) + 1) == ac)
                    i++;
               else
               {
                    write(1, "Fail\n", 5);
                    return (0);
               }
          }
          if (ft_checkmate(av) == 1)
               write(1, "Success\n", 8);
          else
               write(1, "Fail\n", 5);
     }
     else if (ac > 1)
          write(1, "Fail\n", 5);
     else
          write(1, "\n", 1);
     return (0);
}
==
#ifndef _CHECKMATE_H
#define _CHECKMATE_H

# include <unistd.h>
# include <stdlib.h>

size_t    ft_strlen(char *s);
int       ft_opiece(char piece);
int       ft_rook(char **board, int y, int x);
int       ft_pawn(char **board, int y, int x);
int       ft_bishop(char **board, int y, int x);

#endif

==
#include "4-5-check_mate-03.h"
```

```
static void    free_chessboard(char **tab)
{
     int line;

     line = 0;
     while (tab[line])
     {
          free(tab[line]);
          line++;
     }
     free(tab);
}

static char    *ft_strcpy(char *dest, char *src)
{
     int  i;

     i = 0;
     while (src[i])
     {
          dest[i] = src[i];
          i++;
     }
     dest[i] = '\0';
     return (dest);
}

static char    **copy(char *argv[], char **tab)
{
     int i;
     int j;

     i = 0;
     j = 1;
     while (argv[j])
     {
          tab[i] = ft_strcpy(tab[i], argv[j]);
          i++;
          j++;
     }
     tab[i] = NULL;
     return (tab);
}

static int     check_chessboard(char **tab)
{
     int i;
     int j;
     int size;

     i = 0;
     size = ft_strlen(tab[i]);
     while (tab[i])
     {
          j = 0;
          while (tab[i][j])
          {
               if (tab[i][j] == 'R' && check_rook(tab, i, j))
                    return (1);
               if (tab[i][j] == 'B' && check_bishop(tab, i, j,
size))
                    return (1);
               if (tab[i][j] == 'P' && check_pawn(tab, i, j))
                    return (1);
               if (tab[i][j] == 'Q' && (check_rook(tab, i, j) ||
                              check_bishop(tab, i, j, size)))
                    return (1);
               j++;
          }
          i++;
     }
     return (0);
}

int            main(int argc, char *argv[])
{
     char **tab;
     int       i;

     i = 0;
     if (argc != 1)
     {
          if (!(tab = malloc(sizeof(char *) * argc)))
               return (-1);
          while (i < argc - 1)
          {
               if (!(tab[i] = malloc(sizeof(char) * argc - 1)))
                    return (-1);
               i++;
          }
          tab = copy(argv, tab);
          if (check_chessboard(tab) == 1)
               write(1, "Success", 7);
          else
```

```
               write(1, "Fail", 4);
          free_chessboard(tab);
     }
     write(1, "\n", 1);
     return (0);
}
```

==

```
#ifndef CHECK_MATE_H
# define CHECK_MATE_H

# include <stdlib.h>
# include <unistd.h>

int       ft_strlen(char *str);
int       check_rook(char **tab, int i, int j);
int       main(int argc, char *argv[]);
int       check_pawn(char **tab, int i, int j);
int       check_bishop(char **tab, int i, int j, int size);

#endif
```

==

```
#include <unistd.h>
#include "4-5-check_mate.h"

static int     resolve(char **grid, pos kpos, int size)
{
     int  i = 1;

     while (kpos.y-i >= 0)
     {
          if ((UCELL == 'Q') || (UCELL == 'R'))
               return (1);
          else if ((UCELL == 'B') || (UCELL == 'P'))
               break;
          i++;
     }
     i = 1;
     while (kpos.y+i < size)
     {
          if ((DCELL == 'Q') || (DCELL == 'R'))
               return (1);
          else if ((DCELL == 'B') || (DCELL == 'P'))
               break;
          i++;
```

```
}
i = 1;
while (kpos.x-i >= 0)
{
     if ((LCELL == 'Q') || (LCELL == 'R'))
          return (1);
     else if ((LCELL == 'B') || (LCELL == 'P'))
          break;
     i++;
}
i = 1;
while (kpos.x+i < size)
{
     if ((RCELL == 'Q') || (RCELL == 'R'))
          return (1);
     else if ((RCELL == 'B') || (RCELL == 'P'))
          break;
     i++;
}
i = 1;
while (kpos.y-i >= 0 && kpos.x-i >= 0)
{
     if ((ULCELL == 'Q') || (ULCELL == 'B'))
          return (1);
     else if ((ULCELL == 'R') || (ULCELL == 'P'))
          break;
     i++;
}
i = 1;
while (kpos.y-i >= 0 && kpos.x+i < size)
{
     if ((URCELL == 'Q') || (URCELL == 'B'))
          return (1);
     else if ((URCELL == 'R') || (URCELL == 'P'))
          break;
     i++;
}
i = 1;
while (kpos.y+i < size && kpos.x+i < size)
{
     if ((i == 1) && (DRCELL == 'P'))
          return (1);
     if ((DRCELL == 'Q') || (DRCELL == 'B'))
          return (1);
     else if ((DRCELL == 'R') || (DRCELL == 'P'))
          break;
     i++;
     }
     i = 1;
     while (kpos.y+i < size && kpos.x-i >= 0)
     {
          if ((i == 1) && (DLCELL == 'P'))
               return (1);
          if ((DLCELL == 'Q') || (DLCELL == 'B'))
               return (1);
          else if ((DLCELL == 'R') || (DLCELL == 'P'))
               break;
          i++;
     }
     return (0);
}

static void    find_king(char **grid, pos *kpos)
{
     int  x;
     int  y;

     y = 0;
     while (*(grid + y))
     {
          x = 0;
          while (*(*(grid + y) + x))
          {
               if (*(*(grid + y) + x) == 'K')
               {
                    kpos->x = x;
                    kpos->y = y;
                    return ;
               }
               x++;
          }
          y++;
     }
}

static int     check_mate(char **grid, int size)
{
     pos  kpos;

     find_king(grid, &kpos);
     if (resolve(grid, kpos, size))
          return (1);
     return (0);
}
```

```
int  main(int argc, char **argv)
{
     int  i;

     if (argc > 1)
     {
          i = 0;
          while (*(argv + i + 1))
          {
               *(argv + i) = *(argv + i + 1);
               i++;
          }
          *(argv + i) = NULL;
          i = 0;
//        while (*(argv + i))
//        {
//             write(1, *(argv + i), 4);
//             write(1, "\n", 1);
//             i++;
//        }
          check_mate(argv, argc - 1) ? write(1, "Success", 7) :
write(1, "Fail", 4);
     }
     write(1, "\n", 1);
}

==
#ifndef CHECK_MATE_H
# define CHECK_MATE_H

# define UCELL grid[kpos.y-i][kpos.x]
# define DCELL grid[kpos.y+i][kpos.x]
# define LCELL grid[kpos.y][kpos.x-i]
# define RCELL grid[kpos.y][kpos.x+i]
# define ULCELL grid[kpos.y-i][kpos.x-i]
# define URCELL grid[kpos.y-i][kpos.x+i]
# define DRCELL grid[kpos.y+i][kpos.x+i]
# define DLCELL grid[kpos.y+i][kpos.x-i]


typedef struct position
{
     int  x;
     int  y;
}                   pos;
```

```
#endif
```

==./4-6-rostring.txt==

Assignment name : rostring
Expected files   : rostring.c
Allowed functions: write, malloc, free

Write a program that takes a string and displays this string after rotating it
one word to the left.

Thus, the first word becomes the last, and others stay in the same order.

A "word" is defined as a part of a string delimited either by spaces/tabs, or
by the start/end of the string.

Words will be separated by only one space in the output.

If there's less than one argument, the program displays \n.

Example:

```
$>./rostring "abc   " | cat -e
abc$
$>
$>./rostring "Que la      lumiere soit et la lumiere fut"
la lumiere soit et la lumiere fut Que
$>
$>./rostring "     AkjhZ zLKIJz , 23y"
zLKIJz , 23y AkjhZ
$>
$>./rostring | cat -e
$
$>
```

==

```
#include <unistd.h>
#include <stdlib.h>

void ft_putstr(char *str)
{
     int i;

     i = 0;
     while (str[i])
     {
          write(1, &str[i], 1);
          i++;
     }
}

int       main(int argc, char **argv)
{
     char *mot;
     int       i;
     int       d;
     int       k;

     i = 0;
     k = 0;
     mot = NULL;
     if (argc > 1)
     {
          while (argv[1][i] && (argv[1][i] == ' '
                         || argv[1][i] == '\t' || argv[1][i] ==
'\n'))
               i++;
          d = i;
          while (argv[1][i] && argv[1][i] != ' '
                    && argv[1][i] != '\t' && argv[1][i] != '\n')
          {
               k++;
               i++;
          }
          mot = (char*)malloc(sizeof(char) * k + 1);
          i = 0;
          while (i < k)
          {
               mot[i] = argv[1][d + i];
               i++;
          }
          mot[k] = '\0';
          i = d + k;
          while (argv[1][i] && (argv[1][i] == ' '
                         || argv[1][i] == '\t' || argv[1][i] ==
'\n'))
               i++;
          d = 0;
          while (argv[1][i])
          {
               if (d == 1 && argv[1][i] != ' ' &&
                         argv[1][i] != '\t' && argv[1][i] !=
'\n')
               {
                    write(1, " ", 1);
                    write(1, &argv[1][i], 1);
                    d = 0;
               }
               else if (d == 0 && argv[1][i] != ' ' &&
                         argv[1][i] != '\t' && argv[1][i] !=
'\n')
                    write(1, &argv[1][i], 1);
               else
                    d = 1;
               i++;
          }
          if (i > k)
               write(1, " ", 1);
          ft_putstr(mot);
          free(mot);
     }
     write(1, "\n", 1);
     return (0);
}
====
#include <stdlib.h>
#include <unistd.h>

int       ft_nb_char(char *str)
{
     int       i;
     int       cnt;
     int       first;

     i = 0;
     cnt = 0;
     first = 1;
     while (str[i])
     {
          if ((str[i] == ' ' || str[i] == '\t') && first == 1)
          {
               cnt++;
               first = 0;
          }
          else if (str[i] != ' ' && str[i] != '\t')
          {
               cnt++;
               first = 1;
          }
          i++;
     }
     return (cnt);
}

char *trim_begin_end_space(char *str)
{
     char *s;
     int       i;
     int       j;
     int       k;

     i = 0;
     k = 0;
     j = 0;
     while (str[j])
          j++;
     while (str[i] == ' ' || str[i] == '\t')
          i++;
     while (str[j - 1] == ' ' || str[i] == '\t')
          j--;
     s = (char*)malloc(sizeof(char) * (j - i + 1));
     if (s == NULL)
          return (NULL);
     while (k < j - i)
     {
          s[k] = str[i + k];
          k++;
     }
     s[k] = '\0';
     return (s);
}

char *epur_str(char *str)
{
     int       t[] = { -1, 0 };
     int       first;
```

```
    char *s;

    first = 1;
    str = trim_begin_end_space(str);
    s = (char*)malloc(sizeof(char) * (ft_nb_char(str) + 1));
    while (str[++t[0]])
    {
        if (str[t[0]] == ' ' || str[t[0]] == '\t')
        {
            if (first == 1)
                s[t[1]++] = str[t[0]];
            first = 0;
        }
        else
        {
            first = 1;
            s[t[1]++] = str[t[0]];
        }
    }
    free(str);
    s[t[1]] = '\0';
    return (s);
}

void rostring(char *str)
{
    int     i;
    int     j;

    i = 0;
    j = 0;
    str = epur_str(str);
    if (str != NULL)
    {
        while (str[i] != ' ' && str[i] != '\t' && str[i])
            i++;
        i++;
        while (str[i + j])
        {
            write(1, &str[i + j], 1);
            j++;
        }
        if (str[i])
            write(1, " ", 1);
        j = -1;
        while (++j < i - 1)
            write(1, &str[j], 1);
```

```
          free(str);
     }
}

int       main(int argc, char **argv)
{
     if (argc > 1)
          rostring(argv[1]);
     write(1, "\n", 1);
     return (0);
}
```

```
======================================./5-0-
brainfuck.txt=========================================
Assignment name  : brainfuck
Expected files   : *.c, *.h
Allowed functions: write, malloc, free
----------------------------------------------------------
----------------------

Write a Brainfuck interpreter program.
The source code will be given as first parameter.
The code will always be valid, with no more than 4096
operations.
Brainfuck is a minimalist language. It consists of an
array of bytes
(in our case, let's say 2048 bytes) initialized to zero,
and a pointer to its first byte.

Every operator consists of a single character :
- '>' increment the pointer ;
- '<' decrement the pointer ;
- '+' increment the pointed byte ;
- '-' decrement the pointed byte ;
- '.' print the pointed byte on standard output ;
- '[' go to the matching ']' if the pointed byte is 0
(while start) ;
- ']' go to the matching '[' if the pointed byte is not 0
(while end).

Any other character is a comment.
```

```
Examples:

$>./brainfuck "++++++++++[>+++++++>++++++++++>+++>+<<<<-]
>++.>+.+++++++..+++.>++.<<+++++++++++++++.>.++
+.------.--------.>+.>." | cat -e
Hello World!$
$>./brainfuck "+++++[>++++[>++++H>+++++i<<-]>>>++
\n<<<<-]>>--------.>+++++.>." | cat -e
Hi$
$>./brainfuck | cat -e
$
==========================================================
================================
```

```c
#include <unistd.h>

int  main(int argc, char **argv)
{
     char string[2048];
     char *str;
     char *ptr;
     int       i;

     if (argc != 2)
     {
          write(1, "\n", 1);
          return (0);
     }
     i = 0;
     while (i < 2048)
     {
          string[i] = 0;
          i++;
     }
     ptr = *(argv + 1);
     str = &string[0];
     while (*ptr)
     {
          if (*ptr == '>')
               str++;
          else if (*ptr == '<')
               str--;
          else if (*ptr == '+')
               (*str)++;
          else if (*ptr == '-')
               (*str)--;
          else if (*ptr == '.')
```

```
               write(1, str, 1);
          else if (*ptr == '[' && !*str)
          {

               i = 1;
               while (i > 0)
               {
                    ptr++;
                    if (*ptr == '[')
                         i++;
                    else if (*ptr == ']')
                         i--;
               }
          }
          else if (*ptr == ']' && *str)
          {
               i = 1;
               while (i > 0)
               {
                    ptr--;
                    if (*ptr == ']')
                         i++;
                    else if (*ptr == '[')
                         i--;
               }
          }
          ptr++;
     }
}

==
#include <stdlib.h>
#include <unistd.h>

void brainfuck(char *str)
{
     int  tab[2048] = {0};
     int  *ptr;
     int  loop_count;

     ptr = tab;
     loop_count = 0;
     while (*str)
     {
          if (*str == '>')
               ptr++;
          else if (*str == '<')
```

```
               ptr--;
          else if (*str == '+')
               ++(*ptr);
          else if (*str == '-')
               --(*ptr);
          else if (*str == '.')
               write(1, ptr, 1);
          else if (*str == '[' && *ptr == 0)
          {
               loop_count = 1;
               while (loop_count != 0)
               {
                    str++;
                    if (*str == ']')
                         --loop_count;
                    if (*str == '[')
                         ++loop_count;
               }
          }
          else if (*str == ']' && *ptr != 0)
          {
               loop_count = 1;
               while (loop_count != 0)
               {
                    str--;
                    if (*str == '[')
                         --loop_count;
                    if (*str == ']')
                         ++loop_count;
               }
          }
          str++;
     }
}

int       main(int argc, char *argv[])
{
     if (argc == 2)
          brainfuck(argv[1]);
     else
          write(1, "\n", 1);
     return (0);
}
```

===

```
#include <unistd.h>
```

```
#include <stdlib.h>

#define BUFF_SIZE 2048

int            main(int argc, const char *argv[])
{
     int       i;
     int       loop;
     char *pointer;

     if (argc == 2)
     {
          i = 0;
          if (!(pointer = (char *)malloc(sizeof(char) *
BUFF_SIZE + 1)))
               return (-1);
          while (i <= BUFF_SIZE)
               pointer[i++] = '\0';
          i = 0;
          while (argv[1][i])
          {
               argv[1][i] == '<' ? pointer += 1 : pointer;
               argv[1][i] == '>' ? pointer -= 1 : pointer;
               argv[1][i] == '+' ? *pointer += 1 : *pointer;
               argv[1][i] == '-' ? *pointer -= 1 : *pointer;
               if (argv[1][i] == '.')
                    write(1, &*pointer, 1);
               if (argv[1][i] == '[' && !*pointer)
               {
                    loop = 1;
                    while (loop)
                    {
                         i += 1;
                         argv[1][i] == '[' ? loop += 1 : loop;
                         argv[1][i] == ']' ? loop -= 1 : loop;
                    }
               }
               if (argv[1][i] == ']' && *pointer)
               {
                    loop = 1;
                    while (loop)
                    {
                         i -= 1;
                         argv[1][i] == '[' ? loop -= 1 : loop;
                         argv[1][i] == ']' ? loop += 1 : loop;
                    }
               }
```

```
            i += 1;
        }
    }
    else
        write(1, "\n", 1);
    return (0);
}
```

```
======================================./5-1-
print_memory.txt=========================================
Assignment name  : print_memory
Expected files   : print_memory.c
Allowed functions: write
----------------------------------------------------------
----------------------
```

Write a function that takes (const void *addr, size_t size), and displays the
memory as in the example.

Your function must be declared as follows:

```
void	print_memory(const void *addr, size_t size);
```

```
---------
$> cat main.c
void	print_memory(const void *addr, size_t size);

int	main(void)
{
	int	tab[10] = {0, 23, 150, 255,
	              12, 16,  21, 42};

	print_memory(tab, sizeof(tab));
	return (0);
}
$> gcc -Wall -Wall -Werror main.c print_memory.c && ./
a.out | cat -e
0000 0000 1700 0000 9600 0000 ff00 0000 ................$
0c00 0000 1000 0000 1500 0000 2a00 0000 ............*...$
0000 0000 0000 0000                     ........$
```

```
================================================================
====================================
#include <unistd.h>

void print_memory(const void *addr, size_t size)
{
     size_t              i;
     size_t              j;
     unsigned char  *p;
     char                *str;

     str = "0123456789abcdef";
     p = (unsigned char *)addr;
     i = 0;
     while (i < size)
     {
          j = 0;
          while (j < 16 && i + j < size)
          {
               write(1, &str[(*(p + i + j)/16) % 16], 1);
               write(1, &str[*(p + i + j) % 16], 1);
               if (j % 2)
                    write(1, " ", 1);
               j += 1;
          }
          while (j < 16)
          {
               write(1, "  ", 2);
               if (j % 2)
                    write(1, " ", 1);
               j++;
          }
          j = 0;
          while (j < 16 && i + j < size)
          {
               if (*(p + i + j) >= 32 && *(p + i + j) < 127)
                    write(1, p + i + j, 1);
               else
                    write(1, ".", 1);
               j += 1;
          }
          write(1, "\n", 1);
          i += 16;
     }
}
```

```
===
#include <unistd.h>

void ft_putchar(char c)
{
     write(1, &c, 1);
}

void ft_putstr(char *s)
{
     while (*s)
          ft_putchar(*s++);
}

void ft_printhex(int n)
{
     int c;

     if (n >= 16)
          ft_printhex(n / 16);
     c = n % 16 + (n % 16 < 10 ? '0' : 'a' - 10);
     ft_putchar(c);
}

void ft_printchars(unsigned char c)
{
     ft_putchar((c > 31 && c < 127) ? c : '.');
}

void print_memory(const void *addr, size_t size)
{
     unsigned char *t = (unsigned char *)addr;
     size_t         i = 0;
     int            col;
     size_t         tmp = 0;

     while (i < size)
     {
          col = -1;
          tmp = i;
          while (++col < 16)
          {
               if (i < size)
               {
                    if (t[i] < 16)
                         ft_putchar('0');
                    ft_printhex(t[i]);
               }
               else
                    ft_putstr("  ");
               ft_putchar((i++ & 1) << 6);
          }
          col = -1;
          i = tmp;
          while (++col < 16 && i < size)
               ft_printchars(t[i++]);
          ft_putchar('\n');
     }
}
===
#include <unistd.h>

void print_memory(const void *addr, size_t size);

int       main(void)
{
     int  tab[15] = {3700067, 58597, 59111,
                            59625, 60139, 60653, 61167, 61681,
62195, 62709, 63223, 63737, 64251,
                            64765, 65279};

     print_memory(tab, sizeof(tab));
     return (0);
}

==
#include <unistd.h>

void ft_putchar(char c)
{
     write(1, &c, 1);
}

void print_memory(const void *addr, size_t size)
{
     const char *base = "0123456789abcdef";
     size_t i = 0;
     unsigned char *str = (unsigned char*)addr;
     char line[17];
     int nb;
     int j;

     // Until finished with line
     while (i < size || i % 16 != 0)
     {
          if (i < size)
          {
               nb = str[i] / 16;
               ft_putchar(base[nb]);
               nb = str[i] % 16;
               ft_putchar(base[nb]);
               // Store printable characters
               line[i % 16] = (str[i] >= 32 && str[i] <= 126) ?
str[i] : '.';
          }
          // Put space in last line
          else
               write(1, "  ", 2);
          i++;
          if (i % 2 == 0)
               ft_putchar(' ');
          if (i % 16 == 0)
          {
               j = 0;
               while (j < 16)
               {
                    // Keep up with location
                    //(i - 16 == beginning of line) + j place in
line
                    // last line
                    if (i - 16 + j >= size)
                         break ;
                    ft_putchar(line[j++]);
               }
               ft_putchar('\n');
          }
     }
}
```

===

```
#include <unistd.h>

void print_memory(const void *addr, size_t size);

int main(void)
{
    char tab[] = {48,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
    print_memory(tab, sizeof(tab));
    return 0;
}

==
#include <unistd.h>

char *g_base = "0123456789abcdef";

void pc(char c)
{
     write (1, &c, 1);
}

void pa(unsigned char c)
{
     if (c > 31 && c < 127)
          pc(c);
     else
          pc('.');
}

void ph(unsigned char c)
{
     pc(g_base[c / 16]);
     pc(g_base[c % 16]);
}

void pl(unsigned char *tab, size_t beg, size_t max)
{
     size_t    i;
     size_t    end = beg + 16;

     for (i = beg; i < end && i < max; i++)
     {
          ph(tab[i]);
          if (i % 2)     pc(' ');
     }
     for (; i < end; i++)
     {
          pc(' ');
          pc(' ');
          if (i % 2)     pc(' ');
     }
     for (i = beg; i < end && i < max; i++)
          pa(tab[i]);
     pc('\n');
}
```

```
void print_memory(const void *addr, size_t size)
{
     unsigned char *tab;

     tab = (unsigned char *)addr;
     for (size_t c = 0; c < size; c += 16)
          pl(tab, c, size);
}

===
#include <stdio.h>

void print_memory(const void *addr, size_t size);

int main()
{
     int  tab[10] = {0, 23, 150, 255,
                   12, 16,  21, 42};
     print_memory(tab, sizeof(tab));
    return 0;
}

===
#include <unistd.h>

void print_hex(unsigned char m)
{
     char values[16] = "0123456789abcdef";
     char trsl[2] = {0};
     int       i = 1;

     if (!m)
     {
          write(1, "00", 2);
     }
     else
     {
          while (i >= 0)
          {
               trsl[i] = values[m % 16];
               m /= 16;
               i--;
          }
          write(1, trsl, 2);
     }
}
```

```
void print_ascii(unsigned char m)
{
     if (m >= 32 && m <= 126)
          write(1, &m, 1);
     else
          write(1, ".", 1);
}

int       calc_pad(int pos)
{
     int i = 0;
     while (pos % 16)
     {
          pos++;
          i += 2;
     }
     i += i / 4;
     return (i);
}

void print_pad(int i)
{
     while (i > 0)
     {
          write(1, " ", 1);
          i--;
     }
}

void    print_memory(const void *addr, size_t size)
{
     unsigned char  *ptr;
     int                 i = 0;
     int                 count_pass;
     int                 tcpt;

     ptr = (unsigned char *)addr;
     while (i < (int)size)
     {
          count_pass = 0;
          tcpt = i;
          while (tcpt < (int)size && count_pass < 16)
          {
               print_hex(ptr[tcpt]);
               tcpt++;
               count_pass++;
               if (tcpt < (int)size)
               {
                    print_hex(ptr[tcpt]);
                    count_pass++;
                    tcpt++;
               }
               write(1, " ", 1);
          }
          print_pad(calc_pad(count_pass));
          count_pass = 0;
          tcpt = i;
          while (tcpt < (int)size && count_pass < 16)
          {
               print_ascii(ptr[tcpt]);
               count_pass++;
               tcpt++;
          }
          write(1, "\n", 1);
          i += count_pass;
     }
}
====
#include <unistd.h>

void    print_memory(const void *addr, size_t size);

int  main(void)
{
     int  tab[10] = {0, 23, 150, 255,
                   12, 16,  21, 42};

     print_memory(tab, sizeof(tab));
     return (0);
}

===
#include <unistd.h>

void ft_putchar(char c)
{
     write (1, &c, 1);
}

void ft_putascii(unsigned char c)
{
     if (c > 31 && c < 127)
          ft_putchar(c);
     else
          ft_putchar('.');
}

void ft_puthex(unsigned char c)
{
     char tab[16] = "0123456789abcdef";

     ft_putchar(tab[c / 16]);
     ft_putchar(tab[c % 16]);
}

void print_line(unsigned char *str, size_t start, size_t max)
{
     size_t i;

     i = start;
     while (i < start + 16 && i < max)
     {
          ft_puthex(str[i]);
          if (i % 2)
               ft_putchar(' ');
          i++;
     }
     while ( i < start + 16)
     {
          ft_putchar(' ');
          ft_putchar(' ');
          if (i % 2)
               ft_putchar(' ');
          i++;
     }

     i = start;
     while(i < start + 16 && i < max)
     {
          ft_putascii(str[i]);
          i++;
     }
     ft_putchar('\n');
}

void print_memory(const void *addr, size_t size)
{
     unsigned char *str;
     size_t    c;
     str = (unsigned char *)addr;
     c = 0;

     while (c < size)
     {
          print_line(str, c, size);
          c += 16;
     }
}

==
#include <unistd.h>

void print_memory(const void *addr, size_t size);

int  main(void)
{
     int  tab[10] = {0, 23, 150, 255,
                   12, 16,  21, 42};

     print_memory(tab, sizeof(tab));
     return (0);
}

===
#include <unistd.h>

void print_memory(const void *addr, size_t size);

int  main(void)
{
     int  tab[10] = {0, 23, 150, 255,
                       12, 16,  21, 42};

     print_memory(tab, sizeof(tab));
     return (0);
}
```

```
======================================./5-2-
ft_itoa_base.txt=========================================
Assignment name  : ft_itoa_base
Expected files   : ft_itoa_base.c
```

Allowed functions: malloc

Write a function that converts an integer value to a null-terminated string using the specified base and stores the result in a char array that you must allocate.

The base is expressed as an integer, from 2 to 16. The characters comprising the base are the digits from 0 to 9, followed by uppercase letter from A to F.

For example, base 4 would be "0123" and base 16 "0123456789ABCDEF".

If base is 10 and value is negative, the resulting string is preceded with a minus sign (-). With any other base, value is always considered unsigned.

Your function must be declared as follows:

```
char *ft_itoa_base(int value, int base);
```

==

```
#include <stdlib.h>

#define abs(a) (a < 0) ? -a : a
char *g_base = "0123456789ABCDEF";

char *ft_itoa_base(int value, int base)
{
    int negative = (base == 10 && value < 0) ? 1 : 0;
    int size = (negative) ? 3 : 2;
    int temp = value;
    while (temp /= base)
        size++;
    char *res = malloc(sizeof(char) * size);
    res[--size] = '\0';
    res[--size] = g_base[abs(value % base)];
    while (value /= base)
        res[--size] = g_base[abs(value % base)];
```

```
     if (negative == 1)
          res[--size] = '-';
     return res;
}
====
#include <stdlib.h>

int       ft_abs(int nb)
{
     if (nb < 0)
          nb = -nb;
     return (nb);
}

char *ft_itoa_base(int value, int base)
{
     char *str;
     int       size;
     char *tab;
     int       flag;
     int       tmp;
     flag = 0;
     size = 0;
     tab = "0123456789ABCDEF";
     if (base < 2 || base > 16)
          return (0);
     if (value < 0 && base == 10)
          flag = 1;
     tmp = value;
     while (tmp /= base)
          size++;
     size = size + flag + 1;
     str = (char *)malloc(sizeof(char) * size  + 1);
     str[size] = '\0';
     if (flag == 1)
          str[0] = '-';
     while (size > flag)
     {
          str[size - 1] = tab[ft_abs(value % base)];
          size--;
          value /=base;
     }
     return (str);
}
===
#include <stdio.h>
#include <stdlib.h>
```

```
/*
** Usage: a.out 23435453 16
**        a.out 23435453 2
**        a.out 23435453 10
**        a.out 23435453 8
*/

int  ft_atoi(const char *str);
char    *ft_itoa_base(int value, int base);

int       main(int argc, char **argv)
{
     if (argc == 3)
     {
          printf("%s\n", ft_itoa_base(atoi(argv[1]),
atoi(argv[2])));
     }
}

===
#include <stdio.h>
#include <stdlib.h>

/*
** Usage: a.out 23435453 16
**        a.out 23435453 2
**        a.out 23435453 10
**        a.out 23435453 8
*/

int  ft_atoi(const char *str);
char    *ft_itoa_base(int value, int base);

int       main(int argc, char **argv)
{
     if (argc == 3)
     {
          printf("%s\n", ft_itoa_base(atoi(argv[1]),
atoi(argv[2])));
     }
}


==========================================./5-3-
```

```
brackets.txt========================================
Assignment name  : brackets
Expected files   : *.c *.h
Allowed functions: write
-----------------------------------------------------------
----------------------

Write a program that takes an undefined number of strings
in arguments. For each
argument, the program prints on the standard output "OK"
followed by a newline
if the expression is correctly bracketed, otherwise it
prints "Error" followed by
a newline.

Symbols considered as 'brackets' are brackets '(' and ')',
square brackets '['
and ']'and braces '{' and '}'. Every other symbols are
simply ignored.

An opening bracket must always be closed by the good
closing bracket in the
correct order. A string which not contains any bracket is
considered as a
correctly bracketed string.

If there is no arguments, the program must print only a
newline.

Examples :

$> ./brackets '(johndoe)' | cat -e
OK$
$> ./brackets '([)]' | cat -e
Error$
$> ./brackets '' '{[(0 + 0)(1 + 1)](3*(-1)){()}}' | cat -e
OK$
OK$
$> ./brackets | cat -e
$
$>
===========================================================
================================
```

```c
#include <unistd.h>
```

```
int is_bracket(char c)
{
     if (c == 40  || c == 91  || c == 123)
          return (1);
     else if (c == 41 || c == 93 || c == 125)
          return (2);
     return (0);
}

int  match(char a, char b)
{
     if (a == ')')
          return (b == '(');
     else if (a == '}')
          return (b == '{');
     else if (a == ']')
          return (b == '[');
     return (0);
}

int  main(int ac, char **av)
{
     char stack[1024];
     int top = -1;
     int  i = 1;
     int j = 0;
     int  printed = 0;

     if (ac < 2)
     {
          write(1, "\n", 1);
          return (0);
     }
     else
     {
          while (i < ac)
          {
               j = 0;
               printed = 0;
               top = -1;
               while (av[i][j])
               {
                    if (is_bracket(av[i][j]) == 1)
                         stack[++top] = av[i][j];
                    else if (is_bracket(av[i][j]) == 2)
                         if (!match(av[i][j], stack[top--]))
                         {
                              write(1, "Error\n", 6);
                              printed = 1;
                              break ;
                         }
                    j++;
               }
               if (top != -1 && printed == 0)
                    write(1, "Error\n", 6);
               else if (printed == 0)
                    write(1, "OK\n", 3);
               i++;
          }
     }
     return (0);
}

===
#include <unistd.h>
#define BUFF_SIZE (4096)

static int     match_brackets(char a, char b)
{
     return ((a == '[' && b == ']') || (a == '{' && b == '}') \
                    || (a == '(' && b == ')'));
}

static int     check_brackets(char *str)
{
     int       i;
     int       top;
     int       stack[BUFF_SIZE];

     i = 0;
     top = 0;
     while (str[i])
     {
          if (str[i] == '(' || str[i] == '{' || str[i] == '[')
               stack[++top] = str[i];
          if (str[i] == ')' || str[i] == '}' || str[i] == ']')
               if (!match_brackets(stack[top--], str[i]))
                    return (0);
          i += 1;
     }
     return (!top);
}
```

```
int             main(int argc, char *argv[])
{
     int       i;

     i = 0;
     if (argc == 1)
          write(1, "\n", 1);
     while (--argc)
     {
          if (check_brackets(argv[++i]))
               write(1, "OK\n", 3);
          else
               write(1, "Error\n", 6);
     }
     return (0);
}
```

```
======================================./5-4-rpn_calc.txt=========================================
Assignment name  : rpn_calc
Expected files   : *.c, *.h
Allowed functions: atoi, printf, write, malloc, free
--------------------------------------------------------------------------------
```

Write a program that takes a string which contains an equation written in Reverse Polish notation (RPN) as its first argument, evaluates the equation, and prints the result on the standard output followed by a newline.

Reverse Polish Notation is a mathematical notation in which every operator follows all of its operands. In RPN, every operator encountered evaluates the previous 2 operands, and the result of this operation then becomes the first of the two operands for the subsequent operator. Operands and operators must be spaced by at least one space.

You must implement the following operators : "+", "-",

"*", "/", and "%".

If the string isn't valid or there isn't exactly one argument, you must print
"Error" on the standard output followed by a newline.

All the given operands must fit in a "int".

Examples of formulas converted in RPN:

```
3 + 4                   >>    3 4 +
((1 * 2) * 3) - 4       >>    1 2 * 3 * 4 -  ou  3 1 2 * *
4 -
50 * (5 - (10 / 9))     >>    5 10 9 / - 50 *
```

Here's how to evaluate a formula in RPN:

```
1 2 * 3 * 4 -
2 3 * 4 -
6 4 -
2
```

Or:

```
3 1 2 * * 4 -
3 2 * 4 -
6 4 -
2
```

Examples:

```
$> ./rpn_calc "1 2 * 3 * 4 +" | cat -e
10$
$> ./rpn_calc "1 2 3 4 +" | cat -e
Error$
$> ./rpn_calc |cat -e
Error$
==========================================================
================================
```

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int       ft_strlen(char *str)
```

```
{
     int       i;

     i = 0;
     while (str[i] != '\0')
          i++;
     return (i);
}

int       ft_isdigit(char c)
{
     if (c >= '0' && c <= '9')
          return (1);
     return (0);
}

int       is_operateur(char *str)
{
     int       i;

     i = 0;
     if (str[i] == '*' || str[i] == '+' || str[i] == '-' ||
str[i] == '%' || str[i] == '/')
     {
          if (ft_isdigit(str[i + 1]) == 0)
               return (1);
     }
     return (0);
}

long      *rpn_calc(char *str)
{
     long *tab;
     int       i;
     int       j;

     i = 0;
     j = 0;
     if (!(tab = (long*)malloc(sizeof(long) * ft_strlen(str))))
          return (NULL);
     while (str[i] != '\0')
     {
          while (is_operateur(str + i) == 0)
          {
               tab[j] = atoi(str + i);
               j++;
               while (str[i] != '\0' && str[i] != ' ')
```

```
                    i++;
               if (str[i] == '\0')
               {
                    printf("Error\n");
                    return (NULL);
               }
               while (str[i] == ' ')
                    i++;
          }
          if (j < 2)
          {
               printf("Error\n");
               return (NULL);
          }
          if (str[i] == '/')
          {
               if (tab[j - 1] == 0)
               {
                    printf("Error\n");
                    return (NULL);
               }
               tab[j - 2] = tab[j - 2] / tab[j - 1];
          }
          else if (str[i] == '-')
               tab[j - 2] = tab[j - 2] - tab[j - 1];
          else if (str[i] == '+')
               tab[j - 2] = tab[j - 2] + tab[j - 1];
          else if (str[i] == '*')
               tab[j - 2] = tab[j - 2] * tab[j - 1];
          else if (str[i] == '%')
          {
               if (tab[j - 1] == 0)
               {
                    printf("Error\n");
                    return (NULL);
               }
               tab[j - 2] = tab[j - 2] % tab[j - 1];
          }
          j--;
          i++;
          while (str[i] == ' ')
               i++;
     }
     if (j > 1)
     {
          printf("Error\n");
          return (NULL);
     }
     return (tab);
}

int       main(int argc, char **argv)
{
     long *tab;

     if (argc == 2 && argv[1][0] != '\0')
     {
          tab = rpn_calc(argv[1]);
          if (tab != NULL)
               printf("%ld\n", tab[0]);
          return (0);
     }
     printf("Error\n");
     return (0);
}
===
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void check_space(char *s, int *i)
{
     int j;

     j = *i;
     while (s[j] == ' ')
          j++;
     *i = j;
}

void skip_num(char *s, int *i)
{
     int j;

     j = *i;
     while (s[j] >= '0' && s[j] <= '9')
          j++;
     *i = j;
}

int       opt(int a, char op, int b)
{
     int  res;
     res = 0;
     if (op == '+')
          return (a + b);
     if (op == '-')
          return (a - b);
     if (op == '/')
          return (a / b);
     if (op == '%')
          return (a % b);
     if (op == '*')
          return (a * b);
     return (00);
}

int       rpn_calc(char *s)
{
     int  tab[10000];
     int  i;
     int  j;

     i = -1;
     j = 0;
     while (s[++i])
     {
          check_space(s, &i);
          if ((s[i] >= '0' && s[i] <= '9') ||     ((s[i] == '+'
|| s[i] == '-') &&
               (s[i + 1] >= '0' && s[i + 1] <= '9')))
          {
               tab[j++] = atoi(&s[i++]);
               skip_num(s, &i);
          }
          if (((s[i] == '-' || s[i] == '+') && (s[i + 1] == ' '
||
               s[i + 1] == '\0')) || s[i] == '/' || s[i] == '*'
|| s[i] == '%')
          {
               if (j < 1 || ((s[i] == '%' || s[i] == '/') &&
tab[j - 1] == 0))
                    return (printf("Error\n"));
               tab[j - 2] = opt(tab[j - 2], s[i], tab[j - 1]);
               j--;
          }
     }
     if (j != 1)
          return (printf("Error\n"));
     return (printf("%d\n", tab[0]));
```

```
}

int       main(int ac, char **av)
{
     if (ac != 2)
          return (printf("Error\n"));
     rpn_calc(av[1]);
     return (00);
}
====
#include <stdio.h>
#include <stdlib.h>
typedef struct s_op
{
     int status;
     int ans;
}                    t_op;

int stack[256];
int pointer = -1;

int pop()
{
     return stack[pointer--];
}

void push(int num)
{
     stack[++pointer] = num;
}

int isemp()
{
     return pointer == -1;
}

int isspc(char c)
{
     return c == ' ' || c == '\t' || c == '\r'
          || c == '\f' || c == '\n' || c == '\v';
}

int isdig(char c)
{
     return (c >= '0' && c <= '9');
}
```

```c
int isop(char c)
{
     return c == '*' || c == '/' || c == '+'
          || c == '-' || c == '%';
}

t_op *doop(char op)
{
     t_op *res = malloc(sizeof(t_op));
     res->status = 1;
     int num1;
     int num2;
     if (!isemp())
          num1 = pop();
     else
     {
          res->status = 0;
          return (res);
     }
     if (!isemp())
          num2 = pop();
     else
     {
          res->status = 0;
          return (res);
     }

     if (op == '+')
          res->ans = num1 + num2;
     else if (op == '-')
          res->ans = num2 - num1;
     else if (op == '*')
          res->ans = num1 * num2;
     else if (op == '/')
     {
          if (num1 == 0)
               res->status = 0;
          else
               res->ans = num2 / num1;
     }
     else
     {
          if (num1 == 0)
               res->status = 0;
          else
               res->ans = num2 % num1;
     }
     return res;
}

int calc(char *equ)
{
     int i = 0;
     t_op *res;
     while(equ[i])
     {
          while (isspc(equ[i]))
               i++;
          if (isop(equ[i]) && (!equ[i + 1] || isspc(equ[i +
1])))
          {
               res = doop(equ[i]);
               if (res->status == 0)
                    return 0;
               else
                    push(res->ans);
          }
          while (isspc(equ[i]))
               i++;
          if (isdig(equ[i]) || (equ[i] == '-' && isdig(equ[i +
1])))
          {
               push(atoi(equ + i));
               if (equ[i] == '-')
                    i++;
          }
          while (isdig(equ[i]))
               i++;
          i++;
     }
     int ans = pop();
     if (isemp())
          printf("%d\n", ans);
     else
          return (0);
     return (1);
}

int main(int ac, char **av)
{
     if (ac != 2)
     {
          printf("Error\n");
          return (0);
```

```
	}
	else
	{
		if (!calc(av[1]))
			printf("Error\n");
	}
	return (0);
}
```

```
======================================./5-5-
options.txt========================================
Assignment name  : options
Expected files   : *.c *.h
Allowed functions: write
----------------------------------------------------------
----------------------
```

Write a program that takes an undefined number of arguments which could be considered as options and writes on standard output a representation of those options as groups of bytes followed by a newline.

An option is an argument that begins by a '-' and have multiple characters which could be : abcdefghijklmnopqrstuvwxyz

All options are stocked in a single int and each options represents a bit of that int, and should be stocked like this :

```
00000000 00000000 00000000 00000000
******zy xwvutsrq ponmlkji hgfedcba
```

Launch the program without arguments or with the '-h' flag activated must print an usage on the standard output, as shown in the following examples.

A wrong option must print "Invalid Option" followd by a newline.

```
Examples :
$>./options
options: abcdefghijklmnopqrstuvwxyz
$>./options -abc -ijk
00000000 00000000 00000111 00000111
$>./options -z
00000010 00000000 00000000 00000000
$>./options -abc -hijk
options: abcdefghijklmnopqrstuvwxyz
$>./options -%
Invalid Option
==========================================================
================================
```

```c
#include <unistd.h>

#define is_alpha(c) (c >= 'a' && c <='z') ? 1 : 0
enum status{INVALID, HELP, SUCESS};
int  g_mem;

void print_bin(int num)
{
    long r = 1;
    r <<= 32;
    char count = 1;
    while (r >>= 1)
    {
        (r & num) ? write(1, "1", 1) : write(1, "0", 1);
        if (count % 8 == 0 && count != 32)
            write(1, " ", 1);
        count++;
    }
}

int       check_flags(char *str)
{
    unsigned i = 0;
    if (str[i] != '-')
        return (INVALID);
    while (str[++i])
        if (!is_alpha(str[i]))
            return (INVALID);
    i = 1;
    while (str[i])
    {
        if (str[i] == 'h')
            return (HELP);
```

```
          g_mem |= (1 << (str[i] - 'a'));
          i++;
     }
     return (SUCESS);
}

int       main(int ac, char **av)
{
     unsigned i = 1;
     unsigned char status = 0;

     if (ac < 2)
     {
          write(1, "options: abcdefghijklmnopqrstuvwxyz\n", 36);
          return (0);
     }
     else
     {
          while (av[i])
          {
               status = check_flags((av[i]));
               if (status == INVALID)
               {
                     write(1, "Invalid Option\n", 15);
                     return (0);
               }
               else if (status == HELP)
               {
                    write(1, "options: 
abcdefghijklmnopqrstuvwxyz\n", 36);
                    return (0);
               }
               i++;
          }
          print_bin(g_mem);
     }
     return (0);
}

====
#include <unistd.h>

int main(int ac, char **av)
{
     int i = 1;
     int  t[32] = {0};
     int j ;
     if(ac == 1)
     {
          write(1,"options: abcdefghijklmnopqrstuvwxyz\n",36);
          return 0;
     }
     i = 1;
     while (i < ac)
     {
          j = 1;
          if(av[i][0] == '-')
          {
               while(av[i][j] && av[i][j] >= 'a'  && av[i][j] <=
'z')
               {
                    if(av[i][j] == 'h')
                    {
                         write(1,"options:
abcdefghijklmnopqrstuvwxyz\n",36);
                         return 0;
                    }

                    t['z' - av[i][j] + 6] = 1;
                    j++;
               }

               if (av[i][j])
               {
                    write(1,"Invalid Option\n",15);
                    return 0;
               }
               j++;
          }
          i++;
     }
     i = 0;
          while (i < 32)
          {
          t[i] = '0' + t[i];
          write(1,&t[i++],1);
               if(i == 32)
                    write(1,"\n",1);
               else if(i % 8 == 0)
                    write(1," ",1);

          }
```

```
        return 0;
}
```

Piscine C Exam Review
Piscine C Exam Review Solutions

C Intermediate Exam Review